UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



Grado en Ingeniería Informática

# TRABAJO FIN DE GRADO

## SISTEMA DE SEGURIDAD Y PROTECCIÓN DE DATOS INTEGRADO PARA MEDIOS DE ALMACENAMIENTO EXTRAÍBLES EN SISTEMAS LINUX

Alejandro Villegas López
Tutor: Eloy Anguiano Rey

Julio de 2016

# SISTEMA DE SEGURIDAD Y PROTECCIÓN DE DATOS INTEGRADO PARA MEDIOS DE ALMACENAMIENTO EXTRAÍBLES EN SISTEMAS LINUX

Autor: Alejandro Villegas López
Tutor: Eloy Anguiano Rey

**Departamento de Ingeniería Informática**
**Escuela Politécnica Superior**
**Universidad Autónoma de Madrid**

Julio de 2016

**Alejandro Villegas López**
***Sistema de seguridad y protección de datos integrado para medios de almacenamiento extraíbles en sistemas Linux***

**Alejandro Villegas López**
C/ Miguel Torres 19
Colmenar Viejo, Madrid 28770

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

*A los iraníes de la Central Nuclear*

*– Venid a ver la violencia inherente al sistema*
*¡Socorro!¡Socorro! Me están reprimiendo*

*Los Caballeros de la Mesa Cuadrada*.

# AGRADECIMIENTOS

En primer lugar quiero agradecer este trabajo a mis padres por haberme ayudado, motivado y en los peores momentos apoyado, para que haya podido llegar hasta aquí. Quiero destacar el esfuerzo de mi madre para enseñarme a estudiar, y la enorme ayuda prestada por mi padre para definir la idea en la que se basa el trabajo, ya que fue a quién acudí cuando vi la conferencia de Chema Alonso, sorprendido por lo que en ella comentaba.

Por otro lado quiero agradecer a Matthew Dharm su paciencia y su esfuerzo para solucionar mis dudas durante el desarrollo del Driver. Matthew, thanks a lot for your dedication. Your information was very helpfully to my work. Now I see the humor of that joke.

También quiero mencionar en este punto a mi tutor, que desde hace un año ya decidió ayudarme a afrontar este reto al que nos hemos enfrentado, y me ha enseñado muchas cosas interesantes a lo largo de estos años.

No olvidarme de la profesora Idoia Alarcón, y que sin ninguna responsabilidad sobre mí, me ha ayudado con mis dificultades a la hora de redactar un documento de esta envergadura.

Como mención especial a mi tutor del grado Carlos Santacruz, gran profesional que me ha aconsejado muy sabiamente a lo largo de los cinco años que han transcurrido desde que entré a la universidad.

Por último, pero no por ello menos importantes, agradecer enormemente a mis compañeros de carrera los cinco mejores años de mi vida. Jamás podré borrar de mi mente las largas y espumosas tardes de viernes estudiando electromagnetismo en la cafetería, los 2.235.781,5 cafés servidos por nuestros queridos Diego y Lorenzo para aguantar conmigo las largas mañanas, tardes, noches y pesadillas con los termopares, y por supuesto el haberme descubierto las tres leyes de la termodinámica, de las cuales llevo una en mi ordenador para que no se me olvide. Pero sobretodo, lo mucho que les aprecio, y que espero que a la pila de recuerdos que ya compartimos, sigamos añadiendo más cada día.

# Resumen

Los objetivos fundamentales de este trabajo son el desarrollo de un módulo de kernel que actúe como driver de bloques para dispositivos de almacenamiento extraíbles USB (Universal Serial Bus.), y un programa con una pequeña interfaz gráfica para la autenticación de usuarios del sistema. Éste usará las credenciales como firma del propietario del PenDrive almacenándolas en un espacio reservado de su memoria para que pueda ser transportado y contrastado por otra instancia del driver en otro sistema.

El diseño del módulo proporciona completa independencia de los otros drivers USB existentes. Esto le permite ser un complemento más, o eliminando los demás drivers, ser el único capaz de manejar estos dispositivos. Esto dificulta el desarrollo, pero le da al proyecto la funcionalidad principal a desarrollar, que es tanto proteger la información que queda almacenada en el dispositivo, como evitar que se conecten medios no deseados que puedan contener *malware* u otros fines indeseados.

La característica distintiva de este módulo es que el almacenamiento queda vinculado, mediante las credenciales del usuario en el sistema, al ordenador donde esté presente el módulo y las credenciales adecuadas para que no pueda ser utilizado en ningún otro ordenador que no comparta ambas cosas y, por tanto, los datos almacenados sean seguros.

El módulo ofrece otras ventajas al estar desarrollado a nivel de kernel. La primera es que sólo puede ser configurado por usuarios con privilegios de administrador dentro del sistema. La segunda permite identificar el modelo del medio físico para poder discriminarlo del resto de dispositivos similares que se utilicen. Por otro lado al ser un módulo de Linux, existe la posibilidad de recompilar el kernel desde cero incorporando este módulo al núcleo del sistema operativo, evitando que pueda ser eliminado o reconfigurado.

Como complemento, la comunicación entre el dispositivo protegido y el host está cifrada para dificultar la lectura de información utilizando sniffers de USB. Sólo se cifra la comunicación que contenga bloques de información útil.

El programa de autenticación de usuarios proporciona, a través de un fichero, los datos de nombre de usuario y contraseña, para que puedan ser contrastados por el módulo en el momento del montaje del dispositivo. En caso de que esta autenticación no sea correcta el programa dejará un mensaje de error en el fichero de salida y el módulo finalizará su función, dejando completamente inaccesible el contenido de la memoria externa.

Por tanto, el sistema está compuesto por un módulo del kernel de Linux para el control del dispositivo, un programa para la autenticación de usuarios, y unas reglas de manejo de dispositivos para la ejecución automática del *software* de autenticación al insertarse el medio protegido.

Finalmente, las pruebas se han ido realizando en paralelo al desarrollo para verificar en todo momento que la implementación era correcta. Para concluir se han llevado a cabo pruebas del sistema cuyos resultados han demostrado su efectividad y su gran potencial en un futuro.

# Palabras clave

**Driver de bloques, módulo de Linux, Linux, USB, Seguridad**

# Abstract

The primary objectives of this project are the development of a kernel module which would be acting as a block driver for USB removable storage devices, and a program with a small graphical interface for user's authentication of the system. This will use the credentials, like the signature of the owner's PenDrive, storing them in a memory space reserved for it to be transported and contrasted by another instance of the driver on another system.

The design module provides complete independence from other existing USB drivers. This allows it to be an added complement, or eliminating other drivers, being the only one able to handle these devices. This makes difficult its development, but it gives the project the core functionality to be developed, which is both to protect information that is stored on the device, and prevent unwanted media which may content malware or other unwanted purposes.

The distinctive feature of this module is that storage is bound by the user's credentials in the system to the computer where the module is situated, and the appropriate credentials so that it can't be used on any computer that does not share both things, and therefore the stored data is safe.

The module offers other advantages as it is developed at kernel level. The first is that it can only be configured by users with administrator privileges within the system. The second allows to identify the physical model to be able to discriminate it from other similar devices which are used. In addition being a Linux module, it is possible to recompile the kernel from the beginning by incorporating this module to the core operating system, preventing it from being eliminated or reconfigured.

As a complement, the communication between the protected device and the host is encrypted to make it difficult to read information using USB sniffers. Only the communication which contains blocks of useful information is encrypted.

The program of user's authentication provides through a file, the data of the user's name and password, so they can be contrasted by the module when mounting the device. In case that this authentication should not be correct, the program will leave an error message in the output file and the module will finish its function, leaving completely inaccessible the content of external memory.

Therefore, the system consists of a kernel module from Linux for device control, a program for the user's authentication, and some rules regarding its management of the devices for the automatic execution of the authentication software when inserted the protected device.

Finally, tests have been made at the same time as the project's development to verify at all times that the implementation was correct. To conclude, tests of the system have been conducted and the results have demonstrated its effectiveness and great potential in the future.

# Keywords

**Block driver, Linux module, Linux, USB, Security**

# ÍNDICE

# LISTAS

## Lista de códigos



## Lista de figuras

# Lista de tablas

# 1 Introducción

## 1.1. Marco del proyecto

El trabajo que se presenta ha sido desarrollado en su integridad por el estudiante Alejandro Villegas López, así como la idea de dicho proyecto. En la motivación se explica cómo surge esta idea y qué se plantea realizar.

El trabajo consiste en un sistema de seguridad para sistemas Linux aplicado a dispositivos de almacenamiento masivo por USB que proteja tanto al *host* como al medio extraíble, desarrollado a bajo nivel con el fin de tener una alta eficiencia y seguridad.

## 1.2. Motivación

La idea para este trabajo surgió de una conferencia del famoso hacker informático Chema Alonso, titulada Thinking about Security, que tuvo lugar en la Escuela Superior de Informática de la Universidad de Castilla–La Mancha. Esta conferencia puede encontrarse a través del enlace [1].

En ella cuenta cómo se llevó a cabo uno de los mejores ataques de ciber-guerra, conocido como `Stuxnet` [2]. En esta conferencia se cuenta cómo el gobierno de los Estados Unidos, en contra del programa de enriquecimiento de Uranio por su posible uso en armamento nuclear, realizado en Irán, lanzó un ciber-ataque para detener el proceso.

La fabricación de este material tenía lugar en unas instalaciones desprovistas de conexión a internet para mayor seguridad. Por lo tanto, el método para infectar la central fue con un simple PenDrive que en su interior contenía cuatro `0-days`. Dos de los cuales tenían la misión de infectar tanto el ordenador al que se conectaba, como a cualquier otro medio de almacenamiento extraíble, y obtener permisos de administrador en el sistema. Los dos restantes serían los encargados de provocar un fallo informático en el sistema de control de la central.

Con el paso del tiempo, los virus se fueron extendiendo de forma imperceptible, hasta que se logró infectar el PenDrive de uno de los trabajadores de la central, que al ser conectado al sistema informático de la misma, activó los otros dos virus residentes en la memoria del dispositivo. Esto hizo saltar las alertas del sistema de control de la central nuclear modificando un decimal de una variable y obligando a los trabajadores a seguir el protocolo de parada de emergencia, interrumpiendo así el funcionamiento de la central durante décadas.

Al ver este enorme fallo de seguridad, a pesar de que se pudiera considerar como un factor humano, al autor de este trabajo se le ocurrió la idea de crear un driver de comunicaciones USB desde cero, cuyo objetivo fuera establecer un criterio de seguridad, por el cual, todo dispositivo de almacenamiento que se conectara y no lo cumpliera, fuera completamente ignorado por sistema. De este modo es irrelevante qué información o programa hubiera en el medio extraíble ya que el núcleo ignoraría por completo que es un medio de almacenamiento y no podría comunicarse con él de ninguna manera.

Existen otros sistemas que ya protegen los datos que contiene el dispositivo. Este sistema, sin embargo, busca proteger primeramente el equipo al que se conecta el periférico, y en segundo lugar, el dispositivo.

Se decidió que debía implementarse a nivel de núcleo para poder controlar completamente el *hardware*. Esto da un nivel de protección mucho más alto que si se tratara de un programa que bloquee las comunicaciones como si asumiera el papel de un firewall, porque sabiendo que es posible que un programa ajeno adquiera permisos de superusuario, podría detener este bloqueo y acceder libremente. Sin embargo en un módulo de kernel no es tan trivial, porque mientras el driver esté funcionando es imposible detenerlo sin provocar consecuencias desastrosas.

## 1.3. Objetivos

El objetivo por tanto de este trabajo es, por tanto, estudiar el funcionamiento de los dispositivos de almacenamiento masivo por USB en los sistemas Linux y diseñar e implementar un sistema completo con un módulo y un programa de usuario que trabajen conjuntamente para lograr una protección del dispositivo USB y, sobre todo, del *host* al que se conecta.

Los detalles sobre objetivos y funcionalidades, así como la definición del trabajo llevado a cabo para conseguirlos, se explicarán en detalle en los capítulos 3 y 4 respectivamente.

## 1.4. Estructura del documento

En el segundo capítulo, Estado del arte 2, se presenta una descripción del sistema operativo escogido así como de aplicaciones de seguridad que tienen semejanza con este trabajo. En el capítulo 3, Objetivos y funcionalidades, se muestran los objetivos generales y específicos marcados para este trabajo y las funcionalidades previstas para el dispositivo. A continuación, en Definición del proyecto (capítulo 4) se justifica la metodología elegida para este desarrollo y se mencionan las distintas herramientas utilizadas. El capítulo 5, Análisis, identifica los roles de usuario, además de presentar el principal caso de uso para finalizar con el catálogo de requisitos. De aquí se continua con Diseño en el capítulo 6 donde se explica el funcionamiento del sistema y sus partes. En el capítulo 7, Desarrollo, se detalla la compleja implementación del sistema. El capítulo 8, Pruebas y evaluación de resultados recoge, por tipología, los casos de prueba más relevantes junto con su resultado. Finalmente en Conclusiones y trabajo futuro 9 se resumen los objetivos logrados en este trabajo así como lo que ha supuesto para el estudiante para terminar identificando futuras líneas de mejora.

En los 7 anexos de la memoria se pueden encontrar tanto fragmentos de código y estructuras de datos utilizadas para ayudar a la comprensión del trabajo por parte del lector, además de un pequeño manual de programador y demás información útil que se referenciará en las siguientes páginas. Estos anexos no son de obligatoria lectura, su intención es proporcionar al lector más información a la que pueda acudir en caso de que algunos aspectos del trabajo le hagan surgir curiosidad sobre el tema.

# 2

# Estado del arte

## 2.1. Kernel Linux

Linux es un sistema operativo basado en Unix, desarrollado en lenguaje C, a principios de los años noventa por el estudiante de ciencias de la computación Linus Torvalds. Se caracteriza por ser un sistema gratuito y de código abierto con una fuerte comunidad de desarrolladores y por ser un referente del software libre.

Se estima, según el artículo [3], que el 34.6 % de los servicios web y el 96.4 % de los supercomputadores usan este sistema por su estabilidad, flexibilidad, libertad de modificación y la fuerte comunidad de desarrolladores que esfuerzan día a día en mejorarlo.

Para hacerse una idea de la evolución de este sistema, su primera versión (0.01) se lanzó en septiembre del año 1991. La última versión disponible (4.6) lanzada en mayo de 2016 ya tiene más de 16 millones de usuarios.

### 2.1.1. Arquitectura

Linux se desarrolló como un núcleo monolítico híbrido. Un núcleo monolítico es aquel que trabaja en su totalidad en el espacio de núcleo, y en el que sus componentes se van añadiendo como si fueran piedras unas sobre otras formando un monolito. Esta arquitectura requiere que para sumar o restar funcionalidades debe ser recompilado enteramente. Sin embargo Linux, al ser un núcleo híbrido, también tiene la capacidad para añadir y eliminar módulos en tiempo de ejecución combinando así las ventajas de los núcleos monolíticos y de los micronúcleos. Es el diseñador del sistema el que puede decidir qué elementos se incluyen en el núcleo del sistema y cuales pueden ser introducidos de forma modular e incluso si la capacidad de introducirlos existe o no.

En la figura 2.1 se muestra un esquema de la arquitectura del sistema Linux y las distintas partes que componen su núcleo ordenados según la su jerarquía. Esta información y la explicación más detalla de los núcleos monolíticos y micronúcleos se ha obtenido del libro *Professional Linux Kernel Architecture* que se referencia en [4].

Linux ha sufrido un desarrollo muy extenso desde sus comienzos para ampliar su funcionalidad, aumentar la eficiencia, y para la corrección de los errores derivados de estas. Con el propósito de controlar los cambios se diseñó un sistema de control de versiones de cuatro dígitos que se explica más detalladamente en el anexo A.

Para este trabajo más concretamente, se ha estudiado el funcionamiento del sistema Linux para los dispositivos de almacenamiento USB y su funcionamiento junto con el protocolo SCSI (Small Computer System Interface.), como se detalla más adelante en el capítulo 7. Para que el lector comience a hacerse una idea sobre su arquitectura se muestra la imagen 2.2, la cual muestra la arquitectura de funcionamiento de los protocolos USB y SCSI, cuya información bien detallada en el libro *USB Mass Storage* [6].

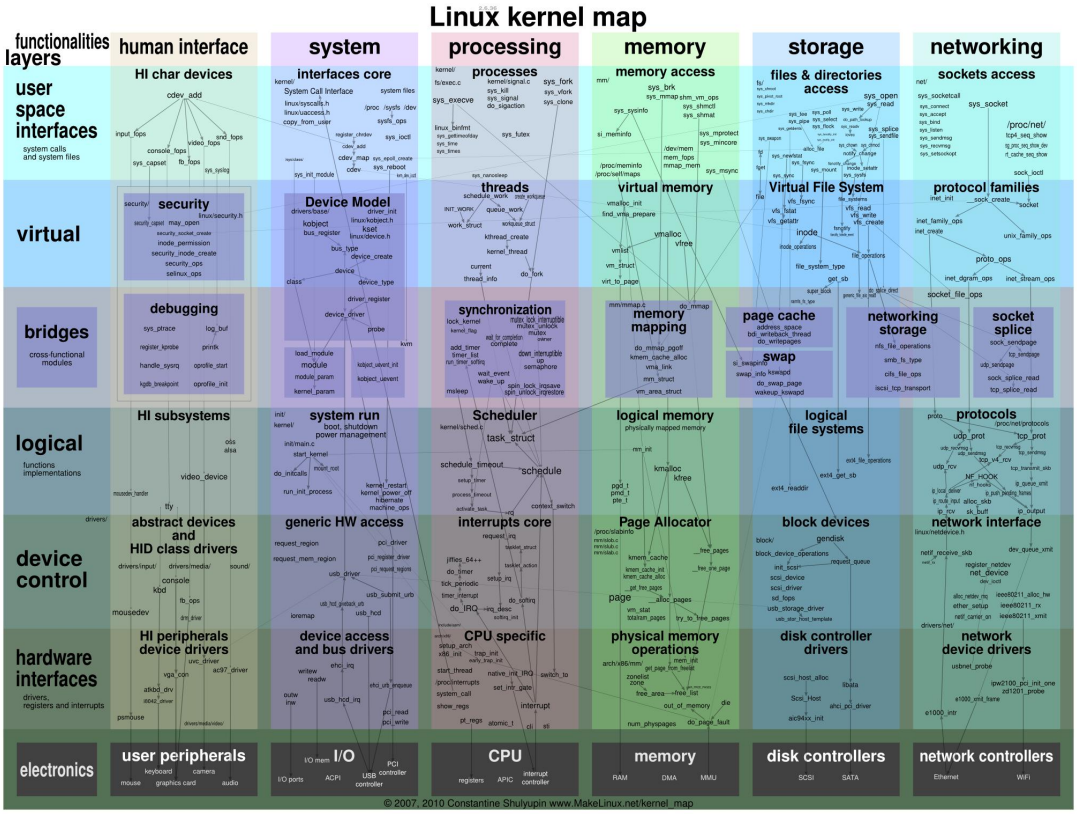


**Figura 2.1:** Mapa de la arquitectura más elemental del kernel de Linux. [5]

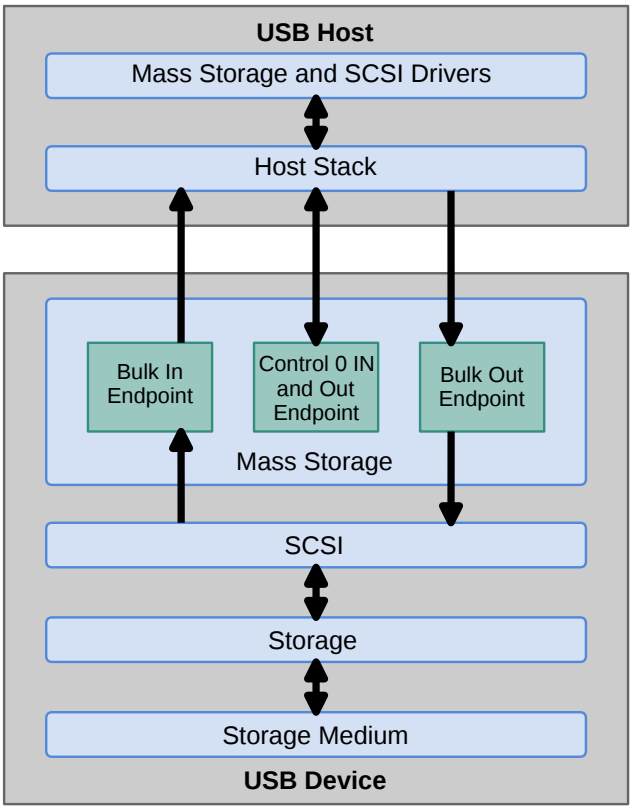


**Figura 2.2:** Esquema de relación entre el protocolo USB y el SCSI en sistemas Linux.

### 2.1.2. Kernel Panic

Al gozar de tanta libertad de modificación, adhesión y eliminación de funcionalidades del kernel, es increíblemente fácil provocar un error en el sistema. Esta facilidad es debida a que al trabajar a nivel de núcleo, se tiene completa libertar de acceder a cualquier parte del *hardware* y de los demás componentes *software*, por lo que el más mínimo error puede desencadenar un fallo crítico en el sistema.

Al disponer de acceso a todo el *hardware* sin restricción el módulo puede sobrescribir la memoria de un programa, pudiendo provocar pérdida de datos y falta de estabilidad. Si embargo no es este el peor problema que se puede producir cuando se comenten errores en desarrollo de módulos. El *Kernel Panic!* es la manifestación más llamativa de estos errores cuando se trabaja a este nivel. Se puede escribir sobre zonas de memoria libre o lo que es peor, sobrescribir parte del núcleo o en zonas de memoria de éste que no se corresponden con las zonas de memoria reservadas para la comunicación entre el módulo y el núcleo. Estos errores que afectan de forma crítica al sistema suelen mostrar, mostrará un característico mensaje `Kernel Panic!`, indicando el motivo del error.

Este mensaje, al igual que todos los procedentes del kernel, son registrados en el fichero de texto localizado en `/var/log/kernel.log` que registra toda esta información, y que es mostrado automáticamente al arranque y cierre del sistema para observar el informe de todas las partes del núcleo durante este proceso. La contabilidad del kernel es esencial a la hora de desarrollar módulos puesto que es la forma más directa de obtener información de lo que sucede con la interacción entre el módulo que se está desarrollando y el núcleo.

## 2.2. Módulo usb-storage

Actualmente en la versión de kernel 3.16.0-4-amd64, que es sobre la que se ha desarrollado el trabajo, se encuentra integrado por defecto en el núcleo del sistema un módulo del kernel para la comunicación con dispositivos de almacenamiento por USB, llamado `usb-storage`. Este driver permite el montaje y la comunicación de cualquier dispositivo de almacenamiento masivo capaz de comunicarse por protocolo USB. Se puede comprobar si está actualmente instalado ejecutando la instrucción que se muestra en el código 2.1.

```
lsmod | grep usb
```

**Código 2.1:** Instrucción para listar los módulos destinados a USB.

Este módulo se añade al núcleo durante el arranque del equipo. Dicho en otras palabras, el núcleo se compiló sin integrar este driver y puede añadirse como funcionalidad extra. Esto permite que el driver pueda ser añadido o eliminado por el administrador cuando sea necesario. El desarrollo de este módulo ha sido llevado a cabo por Matthew Dharm, el cual ha proporcionado asesoramiento, vía e-mail, para el diseño del módulo de este trabajo.

Este driver se apoya en el protocolo SCSI para la creación del sistema de ficheros sobre la memoria del dispositivo y que sea posible navegar entre sus elementos como en cualquier otro almacenamiento. Al haberse desarrollado con licencia GPL (General Public License), el código es de libre acceso, y es posible descargarlo desde los repositorios de Linux, o desde la cuenta de GitHub del desarrollador como se muestra en el enlace [7]. La tecnología por la que funciona este repositorio se explica más adelante en en apartado 4.3.2.

El desarrollo del driver de este trabajo se ha inspirado en el funcionamiento del módulo `usb-storage`. Para ello se ha observado el funcionamiento a través de la herramienta Wireshark, de la que se detalla su uso en la sección 7.6, y con la lectura de determinadas partes de su código para aprender de su comportamiento y cómo deben realizarse determinadas operaciones.

## 2.3. Aplicaciones de seguridad para memorias USB

En la actualdad se dispone de varios productos comerciales para la protección de memorias USB. A continuación se exponen algunos ejemplos representativos. Así mismo se expondrán los elementos diferenciales entre el *software* desarrollado en este trabajo estos productos comerciales.

### 2.3.1. Crypt Setup

`Crypt Setup` es un *software* que ofrece un nivel de protección del contenido de la memoria basado en un particionado cifrado del contenido de su memoria, y una clave maestra que lo descifra. Se apoya en el módulo `DMCrypt` ya disponible en el kernel Linux. Las particiones son construidas por el *software* con `ext4` como particionado base al que se le añade `LUKS` para proporcionar la seguridad.

Para usarlo solo es necesario reparticionar el almacenamiento con este formato. En este momento se establecerá la contraseña y se finalizará la configuración. Puede encontrarse el manual completo de este *software* en el enlace [8].

A diferencia con el sistema que se propone para este trabajo, este *software* se centra exclusivamente en la protección de la memoria del PenDrive, y descuida por completo la seguridad del *host* al que se conecta. Este objetivo es el que se quiere cubrir en la realización de este trabajo.

#### LUKS

LUKS, de sus siglas en inglés, (Linux Unified Key Setup), es una especificación de cifrado de discos para Linux. Su principal objetivo es estandarizar todos los formatos de cifrado no documentados y que causaban graves problemas entre distintos programas. Es compatible incluso en sistemas Windows.

### 2.3.2. Rohos Mini Drive

Esta aplicacion permite crear una partición sobre memorias USB protegida por contraseña para guardar los datos de forma segura. El programa solo es necesario para la creación de dicha partición, por lo que los datos pueden ser escritos y editados en cualquier parte siempre que se conozca la contraseña.

Otras de sus funciones son:

- Creación de una partición virtual cifrada en el espacio libre de una memoria USB.
- Detección automática de la memoria USB y creación automática de la partición cifrada.
- Permite acceder a la partición protegida desde cualquier ordenador ejecutando un programa dentro de la memoria USB y conociendo la contraseña.
- Cifrado automático y rápido.
- La información es cifrada mediante una clave AES de 256 bits.
- El uso de Rohos Mini Drive no requiere permisos de administrador para acceder al PenDrive .
- El teclado virtual protege la contraseña contra programas que registran las teclas pulsadas.
- El tamaño máximo de la partición cifrada: 2 GB.

Sin embargo, Rohos Mini no crea una verdadera partición en la memoria USB, si no que crea una partición virtual que es interpretada por el propio *software*, pero no lo particiona de forma distinta a cualquier dispositivo. El resto de su información se encuentra en [9].

Al igual que en producto `Crypt Setup` 2.3.1, este producto se centra exclusivamente en la protección de los datos alojados en la memoria del dispositivo descuidando la integridad y la seguridad del ordenador al que se conecta.

## 2.4. Otros sistemas de seguridad para USB

Por otro lado, existen memorias USB que integran las medidas de seguridad para la protección de datos en su propio *hardware*. A continuación se presenta un modelo con estas características.

### 2.4.1. Dispositivos securizados

El dispositivo DataTraveler 2000 del fabricante Kingston, que se muestra en la figura 2.3, implementa un cifrado hardware en base al algoritmo AES de 256 bits que bloquea el periférico hasta que se introduce una contraseña por el panel alfanumérico que se encuentra en el mismo dispositivo. Puede verse uno de estos elementos en la figura 2.3.


**Figura 2.3:** Kingston DataTraveler 2000

Sin embargo sigue sin ofrecer protección hacia el *host* del contenido de su memoria, y su precio se incrementa alrededor de un 1350 % sobre dispositivos de las mismas características pero sin protección.

## 2.5. Opiniones y conclusiones

En los productos analizados en la sección anterior, se ha visto que todos los programas de fácil acceso se centran en la creación de una partición protegida por contraseña. Otras alternativas, también estudiadas, aportan un sistema *hardware* que bloquea el dispositivo hasta introducir una contraseña, pero ninguno se centra en la protección del *host* .

Por ello el sistema presentado en este trabajo supone un avance respecto a los productos existentes, ya que amplía el margen de seguridad que existe actualmente en torno a estos dispositivos. Este sistema permite disponer del control del *hardware* USB que puede ser introducido en un sistema con el fin de que no pueda introducir elementos extraños en los *host* implicados.

# 3

# Objetivos y funcionalidades

## 3.1. Introducción

El objetivo principal es crear un sistema formado por dos aplicaciones, una a nivel de usuario y otra a nivel de núcleo, que trabajando conjuntamente protejan el *host* y los dispositivos de almacenamiento extraíbles. El sistema constará de un módulo del núcleo de Linux y una aplicación que emplee las credenciales del sistema para autentificar que el usuario es el propietario del PenDrive. En caso contrario, el dispositivo quedará aislado del sistema por parte del módulo para que sea imposible su comunicación con ninguna parte del equipo.

Como objetivo secundario se pretende que toda la comunicación del *host* con el dispositivo y el almacenamiento de los datos se realice de forma segura, empleando algoritmos de cifrado de bloques que alteren tanto el contenido de los datos efectivos como la tabla de particiones y los registros maestros de la memoria para que no pueda ser leído sin el módulo instalado y sin las credenciales adecuadas.

Otra característica de diseño enfocada a la seguridad e integridad del *host* establece que el usuario debe autenticarse, además de en el sistema, como propietario del PenDrive en base a las mismas credenciales que posea dentro del sistema Linux en el que esté instalado el driver.

## 3.2. Objetivos específicos y funcionalidades

Conocidos ya en rasgos generales las metas propuestas para este trabajo, se procede a enumerar los distintos objetivos derivados de la introducción de este capítulo, en orden según su relevancia, así como las funciones más importantes que realizará el sistema.

### 3.2.1. Objetivos generales

- **Sistema de seguridad:** Proporcionar un sistema de seguridad para sistemas Linux de protección de datos en memorias USB. Esto pretende complementar el resto de productos similares con un propósito parecido y que haya múltiples opciones para que el usuario elija la que más le convenga. Esta alternativa formará parte de la lista de programas de *software* libre donde la comunidad de desarrolladores podrá ayudar a mejorarlo o personalizarlo para sí.
- **Protección del *host*:** Aportar una solución que permita proteger los ordenadores de los PenDrive infectados. Como se ha comprobado en la realización del estado del arte (capítulo 2) de este trabajo, todos los *softwares* de protección de dispositivos de almacenamiento USB se centran en la protección de los datos y no del *host*. Por lo tanto este punto pretende dar una distinción importante

entre el resto de productos existentes y garantizar que el ordenador no pueda ser infectado por un PenDrive. Esto puede ser especialmente útil para equipos que manejen información sensible o que sean elementos críticos de seguridad.

### 3.2.2. Objetivos específicos

- **Desarrollo de un módulo:** Desarrollar desde cero un driver de bloques para el control absoluto de las comunicaciones entre dispositivo y *host* . Este objetivo tiene dos propósitos: el primero es que, al desarrollarse como un módulo del sistema, se tiene control completo del dispositivo sin depender de otro *software* intermedio que pueda aportar fallos de seguridad indeseados y, en segundo lugar, el interés académico que tiene el desarrollo de un driver desde cero para el alumno.
- **Restricción de comunicaciones:** Incomunicar completamente los dispositivos que no cumplan la autenticación de seguridad. Son muchas las formas en las que se puede almacenar *software* malicioso en un PenDrive y al ser extremadamente difícil dar una solución para cada una de ellas y estar al tanto de los nuevos métodos que surgen día a día, si desde el driver se incomunica el dispositivo, no existe otra forma de comunicación. Así se vuelve imposible que se intercambie información o ejecute *software* sobre el *host* . Ni siquiera éste será consciente de tiene un PenDrive conectado a su sistema.
- **Usuarios de Linux :** Emplear el sistema autenticación de usuarios de Linux para garantizar que el usuario del *host* también es dueño del PenDrive. Esto permitirá asegurarse de que el dispositivo sólo pueda ser usado en un ordenador sobre el que el propietario tenga usuario y permisos especiales. Este objetivo principalmente está enfocado a la prevención del robo del dispositivo físico y que no sea posible atacarlo en otro ordenador por un tercero.
- **Compatibilidad:** Asegurar el funcionamiento en paralelo con el módulo `usb-storage` (sección 2.2). Para no perder funcionalidad se pretende implementar de forma que sea compatible con el driver de USB original pudiendo así utilizar cualquier otro medio de almacenamiento que no esté protegido. Con el fin de proteger completamente el *host* se recomienda eliminar el módulo `usb-storage`, pero ese punto queda a elección del administrador del sistema.

### 3.2.3. Funcionalidades

A partir de los objetivos, tanto generales como específicos, listados anteriormente se presentan a continuación las distintas funcionalidades que abarcará el sistema compuesto por los programas `SecSys Authenticator` y `SecSys` que desempeñarán en conjunto para lograr los máximos objetivos posibles.

**F-1.–** Desarrollar `SecSys Authenticator` de tal manera que, utilizando el fichero `/etc/shadow` y la librería Zenity (ver apartado 4.3.1), compruebe que los datos del usuario que se proporcionen a este programa coincidan con los que figuran en el sistema.

**F-2.–** Realizar el registro y el montaje del dispositivo en el sistema desde el driver.

**F-3.–** Almacenar las credenciales en un espacio de memoria reservado del PenDrive para garantizar la autenticidad del dispositivo.

**F-4.–** Leer y procesar la información de los superbloques y tabla de particiones del dispositivo en formato `ext2`.

**F-5.–** Almacenar la contraseña en un espacio reservado de la memoria del dispositivo gestionado por el driver.

**F-6.–** Desarrollar el `SecSys Authenticator` para que se comunique con el driver `SecSys` para autorizar el uso del dispositivo mediante un fichero de texto en la partición `/proc`.

**F-7.–** Gestionar las comunicaciones del dispositivo USB con el *host*.

**F-8.–** Reordenar los drivers actuales para no perder funcionalidad. Para que el sistema de preferencia al módulo desarrollado en este trabajo se deben extraer todos los drivers USB e insertar primero el módulo `SecSys` y después el `usb-storage`.

**F-9.–** Cifrar del contenido del PenDrive en función de la contraseña del usuario propietario del dispositivo.

**F-10.–** Asignar de forma automática el dispositivo protegido al driver desarrollado.

**F-11.–** En caso de que se instale en paralelo al módulo `usb-storage` (sección 2.2), permitir que todos los dispositivos no protegidos sean directamente manejados por éste driver.

**F-12.–** Ejecución automática del `SecSys Authenticator` al conectarse el medio protegido para proceder a su autenticación a través de las *udev*.

**F-13.–** Asegurar el normal funcionamiento de los PenDrive que no sean protegidos siempre y cuando no se haya eliminado el driver `usb-storage`.

# 4

# Definición del proyecto

En este capítulo se detallan el ciclo de vida del software seleccionado y las distintas fases que han tenido lugar a lo largo de la realización del trabajo, tanto en sus fases, como en sus etapas principales y la duración aproximada de las mismas. Por otro lado se explicarán también las herramientas empleadas con una pequeña explicación de cada una y los lenguajes de programación empleados y porqué se han elegido dichas tecnologías.

## 4.1. Metodología

El primer paso antes de comenzar el trabajo fue el estudio de las distintas metodologías de desarrollo de *software* para escoger cuál es la más apropiada para este proyecto teniendo en cuenta que no es una aplicación como tal, ya que como se ha explicado en la introducción (capítulo 1), consta de un programa para la autenticación, y como parte central, un módulo de núcleo del sistema. Un módulo no es un programa sino que es un elemento que añadido al núcleo aporta una nueva funcionalidad al propio núcleo. Esto dificultó la tarea de elección de un ciclo de vida, por lo que se decidió seguir un estilo clásico basado en cascada iterativo, como sigue el esquema de la imagen 4.1, para poder realizar modificaciones a medida que se fuera desarrollando y poder volver atrás en alguna fase cuando se encontrase una dificultad en la realización del driver `SecSys`.

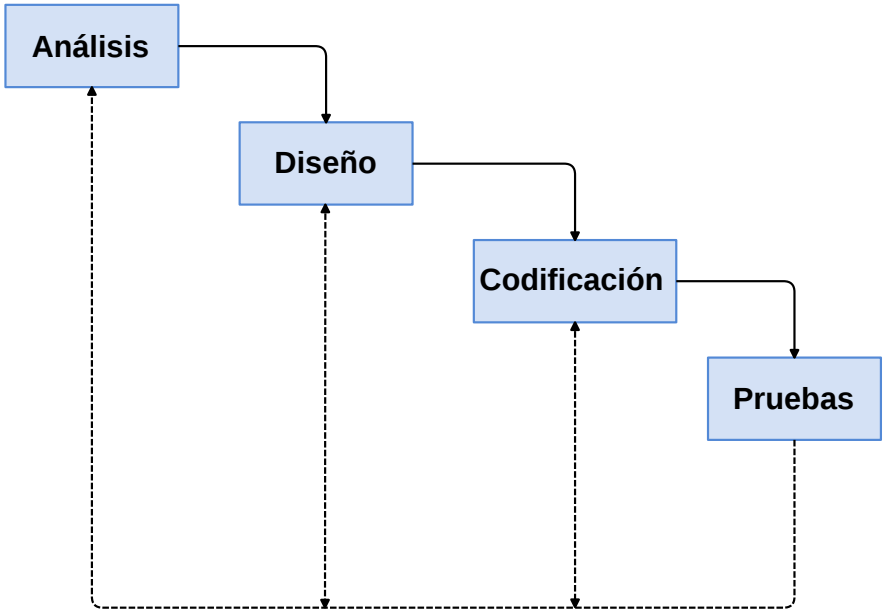


**Figura 4.1:** Esquema del ciclo de vida en cascada iterativo.

### 4.1.1. Ciclo de vida del software

A partir del ciclo de vida en cascada iterativo, cuyo esquema se representa en la imagen 4.1, para la elaboración del sistema `SecSys`, se han extraído de cada una de las fases las tareas en las que consiste cada una de las etapas correspondientes que se enumeran a continuación:

1.– **Análisis:**

- 1.1.– Estudio del estado del arte.
- 1.2.– Investigación y aprendizaje sobre módulos de kernel de Linux.
- 1.3.– Definición de objetivos y funcionalidades.
- 1.4.– Elección del tipo de driver.

2.– **Diseño:**

- 2.1.– Diseño de la comunicación entre el programa `SecSys Authenticator` y el driver `SecSys`.
- 2.2.– Diseño de la aplicación `SecSys Authenticator`.
- 2.3.– Diseño del driver.
- 2.4.– Estudio de las reglas *udev*.

3.– **Desarrollo:**

- 3.1.– Implementación del `SecSys Authenticator`.
- 3.2.– Desarrollo del driver `SecSys`.
- 3.3.– Creación de las reglad *udev*.
- 3.4.– Implementación de la comunicación entre aplicaciones.
- 3.5.– Automatización de la instalación y configuración.

4.– **Pruebas:**

- 4.1.– Pruebas en varios equipos y con varios usuarios de la efectividad del `SecSys Authenticator`.
- 4.2.– Realización de pruebas básicas al `SecSys`.
- 4.3.– Realización de pruebas de compatibilidad al `SecSys` con otros driver.
- 4.4.– Realización de pruebas de estrés al `SecSys`.
- 4.5.– Realización de una batería de pruebas por usuarios ajenos al desarrollo del trabajo.
- 4.6.– Evaluación de los resultados de las pruebas obtenidos en el apartado anterior.
- 4.7.– En base a los resultados, determinar nuevos elementos a desarrollar como trabajo futuro no contemplado en este proyecto.

## 4.2. Planificación

La realización de este trabajo ha tenido lugar desde septiembre de 2015 en cuatro etapas importantes de desarrollo:

- **Etapa 1:** Desarrollo del *software* de autenticación `SecSys Authenticator`.
- **Etapa 2:** Investigación y creación de las reglas *udev*.
- **Etapa 3:** Primera parte del desarrollo del driver `SecSys`. Esta parte se centró en la creación del *driver* de bloques y el registro del PenDrive para que pueda ser usado por el sistema.
- **Etapa 4:** Segunda parte del desarrollo del driver `SecSys`. Esta fase termina con el módulo añadiendo la lectura de la información sobre el estado del dispositivo y comunicación entre el PenDrive y el *host* además de la lectura de las credenciales de seguridad a través de un fichero de texto.

### 4.2.1. Puntos de revisión

A lo largo del proceso han surgido tres puntos de revisión, uno para la aplicación `SecSys Authenticator`, otro para el driver `SecSys`, y como punto final las pruebas de cohesión entre ambas aplicaciones. A continuación se detalla cada punto por separado:

- **Revisión del software `SecSys Authenticator`:** Se comprueba el correcto funcionamiento para varios usuarios y la escritura correcta de las credenciales en el fichero de comunicación con el driver `SecSys`.
- **Revisión del módulo `SecSys`:** Se verifica el correcto montaje del dispositivo y la obtención de las propiedades del mismo.
- **Cohesión de aplicaciones:** Se comprueba que la comunicación entre ambas partes se realiza de forma correcta y sin pérdida de información.

## 4.3. Herramientas utilizadas

Para este trabajo se han seleccionado varias herramientas para el desarrollo de las tres partes que conforman el trabajo, para el control de versiones, la optimización de tareas y otras para ayudar la comprensión del protocolo USB. A continuación se muestra una lista de las herramientas utilizadas:

- Sublime Text 3
- Zenity
- Makefile
- Vim
- Git (BitBucket)
- Git Kraken
- Wireshark
- Ghex
- Bash

- Udev-Rules

A continuación se detallarán las herramientas ya mencionadas, cual es su función y qué han aportado a este trabajo separadas según su tipo.

### 4.3.1. Desarrollo

#### Sublime Text 3

Sublime Text 3 es un editor de texto muy sofisticado y personalizable con versiones para muchos sistemas operativos ya que solo requiere de un interprete de *Python* para ser ejecutado. Debido a su gran flexibilidad ha sido la herramienta para desarrollo de código más empleada en este trabajo.

#### Zenity

Zenity es un *software*, que apoyado en la librería `GTK+`, implementa de forma sencilla unos pequeños diálogos con diversas funcionalidades, como por ejemplo:

- Mensajes informativos, de advertencia o de informe de errores.
- Solicitar nombre de usuario y contraseña.
- Listas.
- Barra de progreso.

Son especialmente útiles para implementar interfaces gráficas sencillas para hacer más vistoso un programa y alejarlo del entorno de la terminal, cosa que suele agradecer un usuario no familiarizado a trabajar con línea de comandos. También da una visión más vistosa a scripts al poner por ejemplo una barra de progreso indicando el tiempo restante para finalizar la ejecución.

#### Makefile

Makefile es una herramienta para la gestión de dependencias. Su uso se centra en la construcción de programas desde el código fuente. En su interior es donde el programador define que objetos deben construirse y cómo. Además aporta una forma rápida y eficiente para muchas tareas, ya que por ejemplo, si detecta que un fichero fuente no ha sido modificado desde la última compilación no lo vuelve a procesar, reduciendo considerablemente el tiempo de ejecución y simplificando la labor de los desarrolladores.

Esta herramienta ha supuesto un alivio de tiempo enorme en la realización de este trabajo porque se requiere de muchas operaciones para una simple ejecución y a veces con comandos largos o duros de recordar. Por esto desde el principio de la etapa de desarrollo se ha automatizado todo el proceso gracias a este *software*, desde la compilación a la ejecución de aplicaciones, filtrado de resultados etc.

#### Vim

Vim, de sus siglas, Vi iMproved, es una versión mejorada de vi. Vi es un editor de texto con 25 años de historia y una gran reputación entre los profesionales de la informática, principalmente desarrolladores de *software* y administradores de sistemas, por ser el uno de los editores de texto con interfaz de terminal más potente por sus múltiples comandos y sobretodo el hacer prescindible el uso del ratón, ahorrando mucho tiempo. Aunque eficaz también es duro de aprender porque cuando se diseñó la distribución de los teclados era algo distinta y

ahora sus atajos conllevan movimientos más costosos. Aún así, se ha empleado mucho este *software* ya que el desarrollador de este trabajo está acostumbrado a su uso en un entorno laboral y tiene ciertos conocimientos sobre su eficacia.

Cabe destacar un pequeño truco asociado a este *software*. Cuando se efectúa una acción de copiado y pegado de código, a menudo es común que los caracteres sufran alteraciones si se han usado codificaciones distintas, o que el propio lenguaje no reconozca algunos símbolos por el mismo motivo. En el compilador de lenguaje C, este error se manifiesta con el mensaje *parásitos en el programa*. Para solucionar este error sólo hace falta abrir el código fuente con este editor y cerrarlo guardando los cambios, porque a pesar de que no hayamos cambiado nada, Vim sólo almacena los caracteres imprimibles, eliminando así los posibles caracteres fuera de la codificación.

### 4.3.2. Control de versiones

#### Git (BitBucket)

BitBucket es un servidor de repositorios Git que además de ser gratuito ofrece privacidad al contenido que se suba a su infraestructura. Proporciona un cliente web y una gran integración con programas de gestión de repositorios Git como Git Kraken, que se explica en el punto 4.3.2.

#### Git Kraken

Es un cliente para repositorios Git con soporte para Windows, Linux e iOS que ofrece una forma muy cómoda y eficiente de manejar varios repositorios simultáneamente. Entre sus características más importantes destacan: el mapa de *commits* con el flujo que sigue el desarrollo, detección automática de los ficheros modificados, fácil gestión de ramas, y una herramienta para resolución de conflictos en la fusión de dos ramas distintas.

### 4.3.3. Estudio

#### Wireshark

Wireshark es un *software* comúnmente empleada en la captura, y procesamiento de tráfico de red. Posee una gran potencia de procesado a tiempo real y múltiples herramientas para filtrado de paquetes, cálculo de estadísticas, interpretación de la información etc.

Se empleó en este trabajo para la comprensión del protocolo USB y para comprobar que el driver `SecSys` desarrollado en este proyecto implementa la comunicación de forma adecuada como se detalla en el apartado 7.6.

#### Ghex

Es un editor de texto en formato hexadecimal permitiendo modificar archivos binarios, y en el caso de este trabajo, ha sido de gran utilidad para la lectura de los superbloque, la tabla de particiones y los paquetes capturados con Wireshark 7.6.

### 4.3.4. Optimización

#### Bash

[10] Es un lenguaje de programación de consola además de un programa que interpreta el propio lenguaje. Se caracteriza por su flexiblidad y eficiencia. Su sintaxis es muy estricta pero permite crear scripts muy rápidamente y con alta eficiencia. Para este trabajo han sido muy útiles varios scripts que automatizan tareas y hacen la labor de desarrollo mucho más fácil y eficiente ahorrando cantidades de tiempo bastante significativas. Entre los scripts para este trabajo se encuentran: para configurar automáticamente el uso de varios monitores, mover ficheros, borrar ficheros temporales, compilación de este documento ...

#### Udev-Rules

Son las reglas para el manejo de dispositivos y son definidas por el administrador del sistema. Se encuentran en el directorio `/etc/udev/rules.d` y figuran con las extensión `.rules`. Para este trabajo se han empleado para implementar un primer filtro de dispositivos no deseados, y en segundo lugar, y el más importante, para activar el *software* `SecSys Authenticator` automáticamente al conectarse el PenDrive protegido.

### 4.3.5. Lenguajes de programación

Desde su origen, el lenguaje empleado para el desarrollo del kernel de Linux ha sido en C. A pesar de tener un desarrollo más lento que en otros lenguajes, C aporta una gran eficiencia y estabilidad. Por eso, C es el único lenguaje en el que se puede desarrollar un módulo de núcleo.

La otra aplicación `SecSys Authenticator` también se ha desarrollado en C para asegurar la compatibilidad en cualquier sistema Linux independientemente de la distribución elegida, ya que todos los sistemas parten del mismo kernel.

También cabe mencionar en este punto el uso del lenguaje *Bash* ya mencionado en el apartado 4.3.4 para la realización de varios scripts que han reducido el tiempo de desarrollo y de pruebas enormemente.

Además se ha empleado el lenguaje LaTeX $2_\varepsilon$ junto con un estilo diseñado por Eloy Anguiano Rey, el tutor de este trabajo, para mejorar la presentación de este documento. Dicho estilo se ha adaptado al formato solicitado para la realización de la memoria según la normativa del centro tanto en cuestión de división de contenido como extensión del documento.

# 5

# ANÁLISIS

## 5.1. Introducción

En este capítulo ya conocidos los objetivos y la estructura básica que se le ha dado al sistema `SecSys` se expondrán los casos de uso y los requisitos en sus dos subcategorías, funcionales y no funcionales, que se han determinado durante la etapa de análisis del proyecto y comenzando la primera etapa del ciclo de vida del software que se muestra en la sección 4.1.1.

## 5.2. Roles de usuario

Existen varios tipos de usuarios que pueden usar el sistema según el rol que ocupen a la hora de ejecutar el sistema `SecSys`. A continuación se exponen estos roles y una breve descripción de cada uno:

- **Usuario sin privilegios:** Este usuario a pesar de ser el dueño del PenDrive no puede editar la configuración del sistema ni tampoco usar la aplicación para permitir el uso de su dispositivo a no ser que un administrador del sistema le de permiso para ejecutar la aplicación `SecSys Authenticator` con permisos de superusuario.
- **Usuario con privilegios:** Este usuario tiene plena libertad para configurar, utilizar o eliminar el sistema `SecSys` de su ordenador. Sin embargo se debe distinguir que no es igual ser administrador del equipo que el propietario del PenDrive que se conecte al *host*. En el caso de no ser el dueño, la comprobación por parte del `SecSys Authenticator` tendría éxito, pero el driver `SecSys` es el encargado de confirmar que el dispositivo es o no de su propiedad.
- **Usuario ajeno al sistema:** Este usuario no puede ni utilizar el sistema `SecSys` ni confirmar que es propietario del dispositivo. Es de este rol de usuario del que se pretende proteger el dispositivo.

## 5.3. Casos de uso

A continuación se muestra el diagrama de casos de uso que englobe los tres roles citados anteriormente y una explicación de las situaciones críticas que se puedan dar en el uso de este sistema. El diagrama se ha divido en dos partes para mayor claridad según si los usuarios son propietarios (imagen 5.1) del dispositivo seguro o no (imagen 5.2).

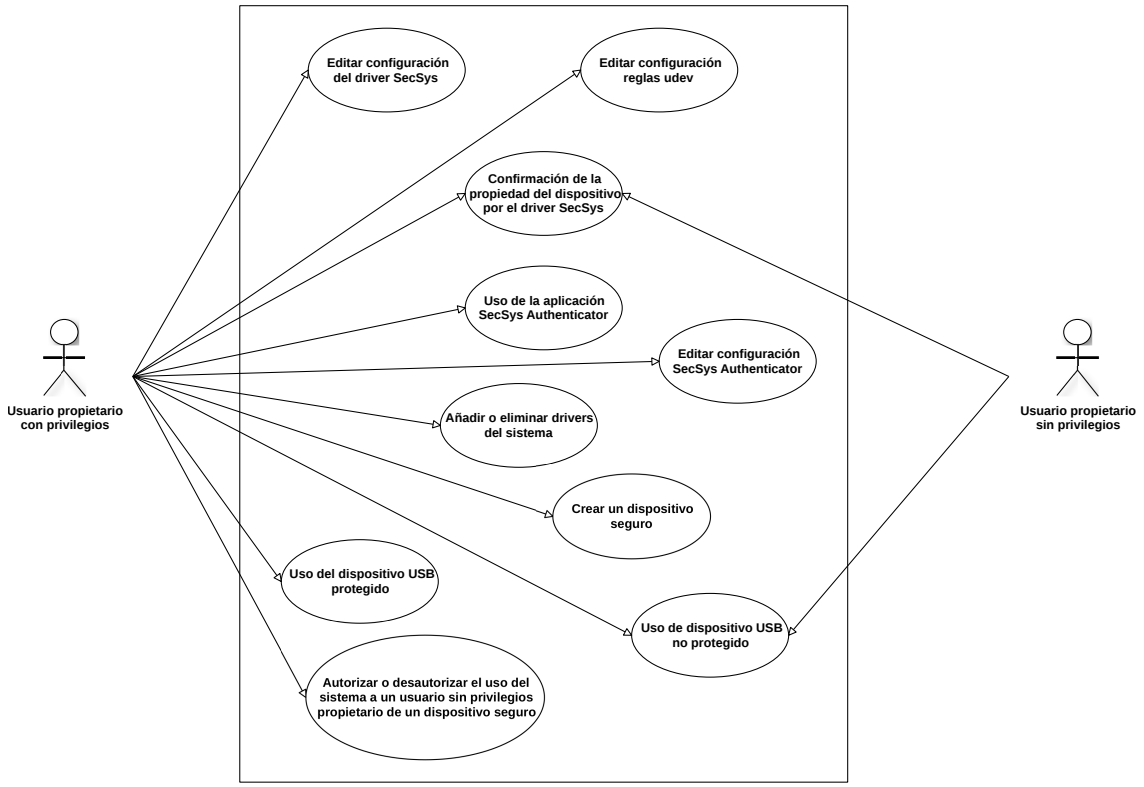


**Figura 5.1:** Diagrama de los casos de uso para los usuarios propietarios del dispositivo seguro.

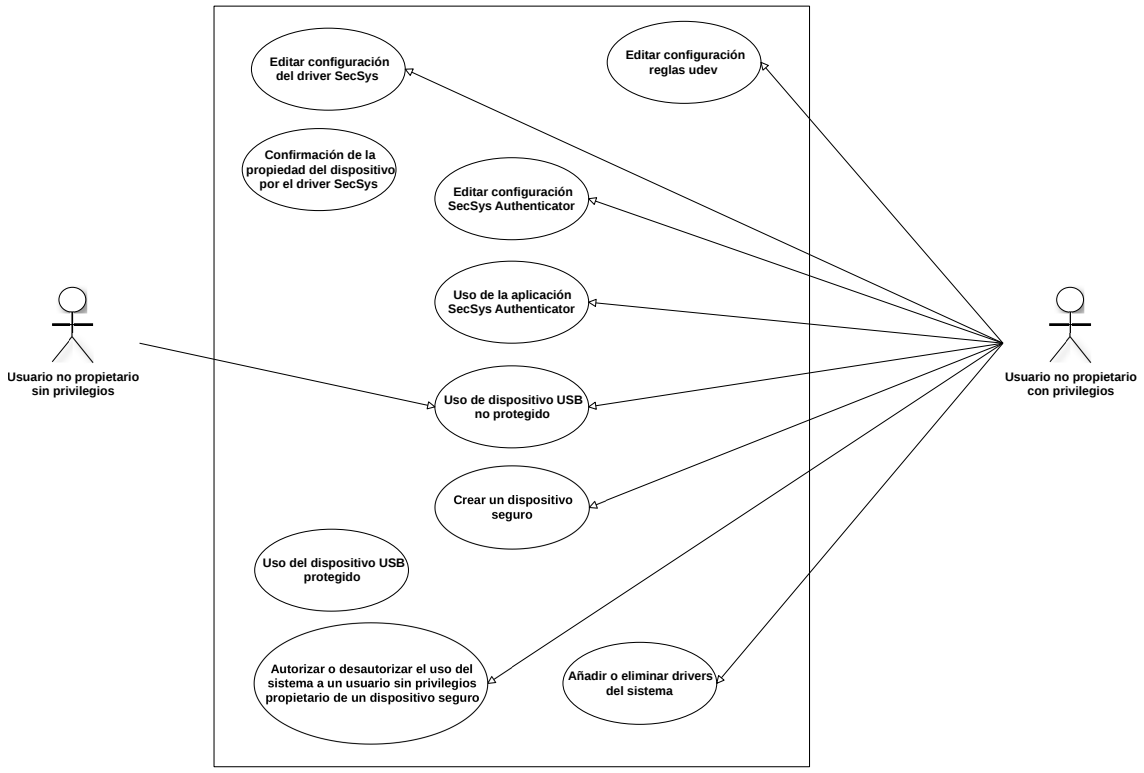


**Figura 5.2:** Diagrama de los casos de uso para los usuarios no propietarios del dispositivo seguro.

## 5.4. Catálogo y definición de requisitos

En esta sección se expondrán los requisitos del sistema de forma detallada separados en función de si se consideran requisitos funcionales o no funcionales.

### 5.4.1. Requisitos funcionales

Esta lista de requisitos detalla las funciones que debe tener el sistema `SecSys` y las acciones que pueden realizar los usuarios de la aplicación.

**RF-1.– Generación de *udev* por scripts:** La aplicación `SecSys Authenticator` para su autenticación en el sistema se ejecutará de forma automática, gracias a la creación automatizada por scripts, gracias a una regla *udev* para la ejecución automática de *software* con el fin de aumentar la usabilidad de cara al requisito final. El usuario que realice esta operación debe poseer permisos de administrador.

**RF-2.– Creación y eliminación de dispositivo seguros:** Los usuarios con privilegios de administrador podrán añadir o eliminar de la lista de dispositivos protegidos cualquier PenDrive y actualizar las configuraciones de todas las partes del sistema sólo por usuarios con privilegios de administrador.

**RF-3.– Gestión de credenciales de seguridad:** Para asegurar la protección del dispositivo y asegurar la identidad del propietario del PenDrive derivan dos requisitos:

- **RF-3.1.– Lectura de las credenciales almacenadas en el PenDrive:** En el espacio de memoria del PenDrive debe almacenarse de forma cifrada los datos de identificación del usuario propietario del dispositivo que sea gestionado por el driver `SecSys` de forma inapreciable por el usuario del sistema.
- **RF-3.2.– Lectura de las credenciales del sistema Linux:** La aplicación `SecSys Authenticator` comprobará que el usuario que emplee el sistema, mediante el fichero `/etc/shadow`, que sus datos de identificación coincidan con los establecidos por el sistema operativo.

**RF-4.– Lectura de ficheros desde el espacio de núcleo:** Como método de comunicación entre `SecSys Authenticator` y el módulo que se instala en el kernel, éste debe poder operar con ficheros desde el espacio de núcleo para el intercambio de información como por ejemplo, las credenciales del usuario o mensajes para variar el comportamiento del driver.

**RF-5.– Configurar driver `SecSys` para responder sólo a dispositivos securizados:** Cada vez que se cree o elimine un dispositivo seguro, el módulo debe ser eliminado, reconfigurado, e insertado de nuevo para que únicamente atienda los PenDrive protegidos y no interfiera en el resto de dispositivos.

**RF-6.– Aplicar algoritmos de cifrado por bloques basados en contraseña:** Al aplicarse a dispositivos de bloques con tamaño fijos se deben emplear algoritmos de cifrado que respeten el tamaño de los datos y que se basen en una frase para realizar el cifrado de forma que éste sea único para cada usuario.

**RF-7.– Lectura y procesamiento de superbloques y tablas de particiones:** El driver `SecSys` debe localizar e interpretar toda la información del superbloque y de la tabla de particiones para

poder interactuar con los ficheros que en él se almacenen, y operar correctamente con el contenido de la memoria del PenDrive.

**RF-8.– Incomunicar dispositivos con credenciales incorrectas:** Los dispositivos que no superen los criterios de seguridad deben ser incomunicados desde el núcleo impidiendo toda comunicación con el dispositivo y ocultarlo de todos los usuarios.

### 5.4.2. Requisitos no funcionales

Los requisitos no funcionales se caracterizan por no especificar funcionalidades concretas para el *software* pero sí sobre sus características del tipo: rendimiento, disponibilidad, accesibilidad, usabilidad, estabilidad, portabilidad, costo, operatividad, interoperabilidad, escalabilidad, concurrencia, mantenibilidad, interfaz y seguridad. A continuación se enumeran dichos requisitos junto con una breve explicación y las categorías anteriormente citadas que abarca cada uno:

**RNF-1.– Protección por credenciales de usuario:** El sistema debe asegurar los dispositivos en base a un nombre de usuario y contraseña para impedir el robo y la suplantación de identidad. Este es un requisito para aportar seguridad al sistema.

**RNF-2.– Empleo de interfaz gráfica para la aplicación de `SecSys Authenticator`:** La aplicación `SecSys Authenticator` , con la finalidad de contar con mejor usabilidad, incorporará una interfaz gráfica a través dela aplicación Zenity explicada en el apartado 4.3.1.

**RNF-3.– Permitir sólo un dispositivo protegido a la vez:** Por restricciones de concurrencia, el driver `SecSys` solo debe permitir su funcionamiento para un dispositivo de forma simultánea ignorando los demás dispositivos mientras ya haya uno conectado al driver.

**RNF-4.– Montaje del dispositivo como unidad de disco externa:** Como requisito de operatividad el dispositivo debe poder accedeserse como un disco duro para la interacción con los datos almacenados en su memoria.

**RNF-5.– Soporte sólo para protocolo USB 2.0:** El driver tendrá una restricción de operatividad sólo para el protocolo USB 2.0 y no a versiones superiores debido al cambio del mismo.

**RNF-6.– Permitir la ejecución de `SecSys Authenticator` para cualquier usuario con permisos:** La aplicación `SecSys Authenticator` requerirá de permisos especiales para su ejecución para poder realizar las comprobaciones de credenciales en los ficheros del sistema.

**RNF-7.– Permitir funcionamiento con otros drivers USB:** El módulo `SecSys` debe aportar interoperabilidad con otros drivers de tal forma que no interfiera con ellos al conectar dispositivos no securizados.

**RNF-8.– Procesamiento y montaje de sistemas de ficheros en Ext2:** Por restricciones de diseño, el módulo sólo manejará sistemas de ficheros particionados en formato Ext2.

**RNF-9.– Velocidad de ejecución del driver:** El módulo debe desarrollarse de forma que aún rompiendo el paradigma de programación estructurada por un requisito de alto rendimiento y estabilidad para el núcleo en el que se integra.

# 6

# Diseño

En este capítulo se expondrá en primer lugar el diseño elegido para el sistema `SecSys` con sus tres elementos principales y las relaciones que existen entre ellas. También irán representadas las comunicaciones e interacciones entre los elementos en un esquema que engloba todo el diseño. Seguidamente se explicará brevemente cada aplicación con sus puntos clave detallando más en profundidad todos los elementos y componentes principales de cada uno.

## 6.1. Estructura de aplicaciones

El trabajo se divide en tres elementos principales, dos de ellos implementados como aplicaciones con una pequeña distinción ya que una de las aplicaciones, concretamente el driver `SecSys` es un módulo del núcleo Linux que amplía su funcionalidad y no una aplicación en sí.

En primer lugar se desarrolló la aplicación `SecSys Authenticator`. Esta aplicación tiene la misión de, utilizando el sistema de usuarios ya implementado por Unix, verificar que el usuario que introduzca sus credenciales en este *software* figure en el registro de usuarios del sistema operativo y que su nombre de usuario y contraseña sean correctos. Una vez comprobada su autenticidad, se escribirán sus datos en un fichero que será interpretado por el módulo `SecSys` para continuar con el proceso a bajo nivel.

En segundo lugar está el driver `SecSys` desarrollado a nivel de núcleo como un módulo del kernel para el sistema. Este es el encargado de comprobar por segunda vez las credenciales de usuario comunicadas por `SecSys Authenticator` con las que se almacenan en el PenDrive para confirmar que es el propietario del PenDrive y de permitir o bloquear la comunicación del dispositivo con el sistema. En caso de que esta verificación no concuerde se detendrá la ejecución del driver `SecSys` incomunicando el dispositivo. En caso de que sea correcto, se procederá al montaje del dispositivo, a la extracción de datos para este fin y en cifrar o descifrar el flujo de datos que se efectúe.

Como tercer elemento están las reglas *udev*. Estas permiten, entre otras cosas, la ejecución de *software* en el momento de la detección de un periférico. Estas son las encargadas de ejecutar automáticamente el programa `SecSys Authenticator` al conectar un dispositivo con el `Vendor` y el `Product` correspondientes al medio que se ha securizado por este *software*.

Se eligió este diseño por varios motivos. En primer lugar, al estar desarrollado como un driver, se asegura la máxima efectividad en la protección del *host* , ya que un módulo posee libertad absoluta para manipular el sistema, y es la primera barrera software que se puede introducir en la comunicación del PenDrive con el *host*. La aplicación `SecSys Authenticator` se definió en el espacio de usuario para que sea más sencilla su implementación y poder añadir una pequeña interfaz gráfica para que sea más cómoda de usar. La comunicación entre estas aplicaciones no supone un problema ya que tanto desde el espacio de usuario como el de núcleo se puede interactuar con ficheros. La integración con las reglas *udev* se ha añadido tanto como por interés

académico como para agilizar el montaje del dispositivo y sea más sencillo de usar de cara al usuario.

En la figura 6.1 se puede observar un esquema detallado del funcionamiento de la aplicación así como la interacción entre las distintas partes que conforman el sistema. En esta figura se muestra la situación de que sólo exista este driver, debido a que es el escenario que de verdad aporta la máxima seguridad al *host*.

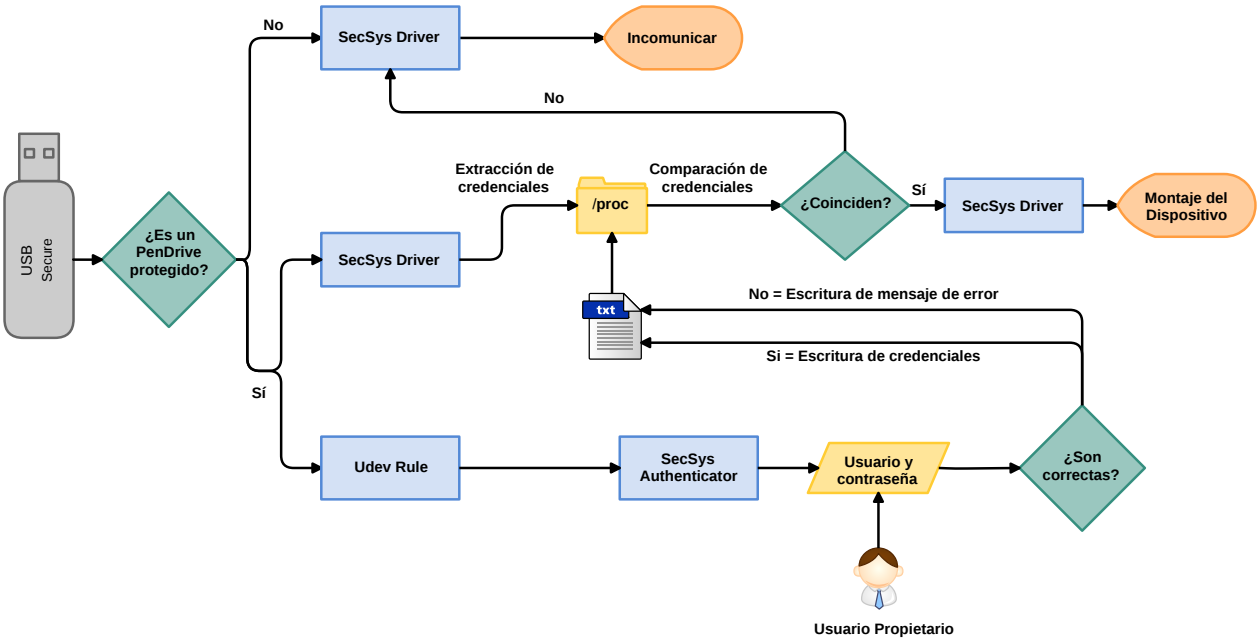


**Figura 6.1:** Esquema de funcionamiento del sistema `SecSys` incluyendo las interaciones entre todas las partes del sistema.

En los siguientes apartados se procederá a describir en detalle cada una de las partes del sistema.

## 6.2. SecSys Authenticator

La aplicación `SecSys Authenticator` se encarga de cotejar las credenciales de usuario del sistema Linux que deberían serán idénticas para el PenDrive. Los datos referentes al usuario se obtienen a través del fichero `/etc/shadow` en el cual el sistema almacena todos los datos para la identificación de usuarios, contraseñas, algoritmos de cifrado, semilla etc.

Tras el tratamiento de la información de este fichero y la comprobación de las credenciales, el programa almacena el resultado en un fichero localizado en la partición `/proc`que está destinada al intercambio de información entre programas de espacio de usuario y espacio de núcleo. Una vez que las credenciales ya hayan sido comprobadas, se cotejan una segunda vez por el driver antes del montaje. El resultado de la autenticación citada anteriormente variará en función de si la comprobación tuvo éxito o no. En el caso de que los datos de usuario no sean correctos, se registra en el fichero para la comunicación un mensaje de error que se interpretará como que el dispositivo no debe ser registrado, indicando al driver que el dispositivo no debe ser montado. En caso afirmativo, se escribirán los datos de igual manera que en fichero `/etc/shadow` para que puedan pasar a ser comprobados por el módulo `SecSys`. Se escogió usar un fichero distinto al de las contraseñas del sistema para proteger su integridad y para que el que se usara con este fin esté debidamente situado en la partición correcta y no en la de almacenamiento de configuración.

## 6.3. SecSys Driver

El módulo `SecSys` está desarrollado desde cero como un driver de bloques para el kernel de Linux en su versión `3.16.0-4-amd64`.

En una primera versión del mismo se empezó a desarrollar como driver de caracteres, pero tiempo después durante el estudio de documentación sobre el desarrollo de drivers USB se descubrió que a pesar de que una buena parte de los dispositivos USB se manejan por drivers de este tipo, los medios de almacenamiento masivo debían ser tratados como dispositivos de bloques para poder actuar con los comandos descritos en la sección 7.5 y que la transmisión de datos fuera a través del protocolo SCSI, para poder construir y manejar sistemas de ficheros sobre la memoria del PenDrive.

Una vez confirmado este diseño con el autor del driver actual para estos dispositivos `Matthew Dharm` vía e-mail, se observó que este tipo de driver tiene predefinidas una serie de operaciones básicas y obligatorias para su uso por parte del núcleo:

- **Probe:** Encargada del registro del PenDrive en el sistema como una unidad de disco.
  En el módulo se define como: `usb_secsys_register_device`.
- **Disconnect:** Encargada de eliminar el dispositivo del sistema para que pueda ser desconectado.
  En el módulo se define como: `usb_secsys_unregister_device`.
- **IOCTL:** Encargada de realizar operaciones de entrada/salida para el dispositivo.
  En el módulo se define como: `secsys_blkdev_ioctl`.
- **GetGeo:** Encargada del paso de información desde el kernel hasta el espacio de usuario.
  En el módulo se define como: `secsys_getgeo`.

Estas funciones siguen un prototipo muy estricto porque de no ser así el núcleo podría no saber con qué argumentos debe llamar a estas funciones. Los prototipos de todas las funciones usadas en este módulo se adjuntan en el anexo C. El desarrollo de las funciones y las operaciones que lleve a cabo, dependen totalmente del programador y es su responsabilidad que el código con el que se construyan no provoquen errores de ejecución, de manejo del dispositivo o se causen inestabilidades en el sistema.

Sin embargo no todas estas funciones se definen al mismo nivel. El primer paso es crear una estructura de tipo `static struct usb_driver` como se muestra en el código 6.1.

```
static struct usb_driver secsys_driver = {
    .name = DRIVER_NAME,
    .id_table = secsys_table,
    .probe = usb_secsys_register_device,
    .disconnect = usb_secsys_unregister_device,
};
```

**Código 6.1:** Definición de la estructura de un driver USB.

Como se puede apreciar a simple vista, en esta estructura sólo se definen dos de las cuatro funciones importantes destacadas con anterioridad. Las funciones restantes con relevancia para el driver se definen en la estructura definida por el desarrollador `SecSys_Data`. Esta estructura se puede ver en el anexo B.

Las otras dos funciones de definen al mismo tiempo en el que el driver comienza el registro del PenDrive en el sistema. Esto es llevado a cabo en la función `usb_secsys_driver_init` ya que en este proceso se debe rellenar los campos de la estructura `fops (File Operations)` como se muestra en el código 6.2.

```
ssd.fops.owner = THIS_MODULE;
ssd.fops.ioctl = secsys_blkdev_ioctl;
ssd.fops.getgeo = secsys_getgeo;
```

**Código 6.2:** Asignación de las funciones para el funcionamiento del driver.

Llegado al punto en el que el sistema ya cuenta con la información mínima necesaria para poder operar el módulo que se inserta para desempeñar el papel de driver de bloques. Se puede ver cuando el sistema ejecuta cada función añadiendo un mensaje al comienzo de cada una de ellas para que lea en el log del núcleo y poder entender mejor el propósito de cada una.

## 6.4. SecSys Udev Rule

El uso de este recurso se decidió emplear en un primer momento para evitar que los dispositivos que no figurasen en la lista de protegidos fueran bloqueados directamente por una regla escrita por el administrador, ya que es el único con permisos para modificar estas reglas. Más adelante, durante el desarrollo del driver, se descubrió que el propio módulo podía ser sensible sobre a qué dispositivos debía ejecutarse, por lo que esta utilidad pareció dejar de tener importancia en el sistema. Finalmente, al ya conocerse su utilidad y funcionamiento, se emplearon para que al insertar un dispositivo securizado se ejecutara automáticamente el `SecSys Authenticator` y fuera más rápido y sencillo el proceso de desbloqueo para el usuario final. Un ejemplo de estas reglas para ignorar los dispositivos no deseados se muestra en el código 6.3.

```
# If a device is NOT a Kingston drive, ignore it.
SUBSYSTEMS=="usb", DRIVERS=="usb", ATTRS{idVendor}!="13fe",
    OPTIONS+="ignore_device"
```

**Código 6.3:** Udev Rule para el sistema SecSys.

Para entender los campos se define a continuación el significado de esta regla en lenguaje formal: *Si el dispositivo conectado atiende a un driver USB dependiente del sistema USB cuyo número* `Vendor` *no coincida con el valor 0x13FE, ignora tal dispositivo.*

Para cumplir la función deseada en este sistema se debe modificar la regla para que ejecute el programa `SecSys Authenticator`, pero esto se explica con detenimiento en la sección 4.3.4.

# 7 DESARROLLO

Ya comprendido el diseño del sistema que se presenta en este trabajo y el diagrama de la estructura y el funcionamiento de `SecSys` en la figura 6.1, en este capítulo se explica el desarrollo completo del trabajo compuesto por el `SecSys Authenticator` y el módulo `SecSys`.

## 7.1. SecSys Authenticator

`SecSys Authenticator` es un programa destinado a confirmar la autenticidad del usuario que usa el sistema de protección que conforma este trabajo. Los usuarios del `SecSys Authenticator` deben figurar como usuarios del propio sistema donde se usa el dispositivo securizado. Este requisito se debe, en primer lugar a la necesidad de asegurar que el usuario que utilice el sistema `SecSys` trabaje desde un ordenador en el que tenga acceso con privilegios de superusuario, y en segundo lugar, al interés que tiene estudiar el sistema de usuarios de Linux y como almacena sus credenciales.

El programa consta de cinco ficheros de código C con sus respectivas bibliotecas:

1.– **checkpass.c:** Código básico del funcionamiento del programa.

2.– **shadow.c:** Código para el parseo del fichero de usuarios de Unix.

3.– **user_unix.c:** Código para el tratamiento de los datos de usuario.

4.– **errors.c:** Código para el informe de errores y su registro en el log propio de la aplicación.

5.– **types.h:** Definiciones útiles en las que se apoya el programa.

Para aumentar la usabilidad de este programa, se ha empleado la aplicación Zenity para crear interfaces gráficos en scripts explicada en el apartado 4.3.1. Esta utilidad permite el uso de interfaces gráficas sencillas. Éstas se han empleado para que el usuario introduzca los datos solicitados de forma cómoda y sencilla. El valor introducido en los diálogos de esta librería se devuelven como retorno del comando que ejecuta el propio diálogo, pudiéndose recuperar esta información por línea de comandos ejecutando la instrucción 7.1, o poniendo la llamada como argumento del programa `SecSys Authenticator` 7.2.

```
zenity --title='SecSys 1.0' --text='Introduzca su contraseña' --password
```

**Cuadro 7.1:** Ejecución de `SecSys Authenticator` con Zenity.

```
./secSystem $(mkpasswd -m sha-512 $(zenity --title='SecSys 1.0' --text='Introduzca su contraseña' --password) %s) %s"
```

**Cuadro 7.2:** Ejecución de `SecSys Authenticator` con Zenity.

### 7.1.1. Log

Existen muchos errores posibles en el tiempo de ejecución de un programa informático y más aún si ese programa interviene en ficheros manejados por el sistema operativo. Por ello, se tomó la decisión de implementar un sistema de log propio, con el fin de separar la información emitida por la aplicación, aportar un medio de depuración y no sobrecargar el que usa el sistema para el resto de aplicaciones.

Este log consta de dos tipos de mensajes, de información y de error. Ambos están encapsulados en una única función que recibe el código del error, una bandera para indicar el tipo del mensaje, y la línea y el fichero desde donde se produjo el fallo, para facilitar la búsqueda de errores. El código de error indica a una función cuál es el mensaje correspondiente para incorporarlo como parte del mensaje que se registrará.

Puede elegirse si la información se muestra por la misma terminal, o guardándose a un fichero localizado por defecto en `/var/log/secsys.log`. En una ejecución normal toda la salida se guarda en este fichero, mientras que si se define la variable `DEBUG_MODE` al compilar, los mensajes se mostrarán por pantalla. Esta última opción puede ser la más conveniente si se precisa ver el avance de los mensajes durante la ejecución.

### 7.1.2. Sistema de usuarios de Linux

El primer paso para la realización de este *software* fue el estudio del fichero `/etc/shadow`. En este fichero se almacenan todos los usuarios del sistema, sus credenciales, y otros datos sobre sus características. A continuación se muestra un ejemplo para su análisis en el cuadro 7.3. Como apoyo a la búsqueda de usuarios, el programa también realiza búsquedas de los nombres de usuario en el fichero `/etc/passwd`.

```
peripocha:$6$7gQyuiyV$b3WrGMULb2QfygUSAd19sbuWy.18Uw5lN9yaF33
xrOhGRVmmoF3Drc6kXxniagReTwKPBCu8YTWz5qMoK.V8p/:16837:0:99999:7:::
```

**Cuadro 7.3:** Entrada de registro de usuario.

El primer parámetro que se observa es el nombre de usuario hasta el carácter ":". El segundo parámetro, situado entre los dos primeros símbolos de dólar, es un número que representa con qué algoritmo se ha cifrado la contraseña. Los posibles valores y sus algoritmos asociados son:

- **MD5:** Código 1.
- **BlowFish:** Código 2a.
- **BlowFish Hand:** Código 2y.
- **SHA de 256 bits:** Código 5.
- **SHA de 512 bits:** Código 6.

El tercero, entre los dos siguientes símbolos de dólar, es la semilla de la función *hash*. Este valor es muy importante porque por cada ejecución de la función de cifrado de la contraseña, puede tomarse una semilla distinta generando múltiples resultados distintos para la misma contraseña, haciendo así imposible su comprobación en un futuro. Como último parámetro de interés para este trabajo, pero no último del registro de usuario, está finalmente la contraseña separada por el símbolo dólar que finaliza la semilla, hasta la aparición nuevamente del carácter ":". En este punto ya se cuenta con información suficiente para contrastar los datos que introduzca el usuario al programa `SecSys Authenticator` con los registrados por el sistema operativo. Cabe destacar que para proteger la identidad del usuario, las contraseñas se guardan cifradas con uno de los algoritmos unidireccionales anteriormente citados, y la única forma de compararlas es cifrando la que se introduzca al `SecSys Authenticator` y comprar su resultado.

En el cuadro 7.4 se presenta un ejemplo de cómo generar la contraseña en base a los datos introducidos al programa, y la información correspondiente al usuario (algoritmo de cifrado, semilla y contraseña original).

*mkpasswd -m sha-512 __CONTRASEÑA__ __SEMILLA__*

**Cuadro 7.4:** Construcción de la contraseña de usuario para poder contrastarla.

### 7.1.3. Ejecución

La ejecución del programa `SecSys Authenticator` sigue este orden. El primer paso que realiza es la comprobación de que dispone de acceso al fichero `/etc/shadow`. En este punto es donde se finalizaría si el usuario que ejecute este programa no tuviera permisos de lectura sobre el fichero. En segundo lugar utilizando, además de dicho archivo, el fichero `/etc/passwd`, que no requiere permisos especiales de lectura, para comprobar que el nombre de usuario introducido corresponde a un usuario auténtico. En tal caso se procede al tercer paso que es la solicitud de la contraseña, a la obtención de los parámetros del usuario, y a la comprobación de autenticidad de la contraseña como se mostró anteriormente en el cuadro 7.4.

En las próximas secciones se procederá a explicar con carácter general, qué son y cómo funcionan los drivers en Linux, y cómo se ha desarrollado el módulo del kernel para éste proyecto.

## 7.2. ¿Qué es un driver?

Antes de comenzar con la descripción del desarrollo del driver `SecSys`, se procederá en esta y en la siguiente sección a definir el concepto de driver y aspectos relacionados básicos para comprender la explicación.

Un driver, o controlador de dispositivo, es un programa informático que permite la comunicación entre el sistema operativo y un periférico, ofreciendo una interfaz para operar con su hardware sin que el sistema operativo tenga que conocer cómo funciona cada dispositivo en concreto. Se muestra un esquema explicativo de la función de un driver en la figura 7.1.

Estos programas suelen destinarse a un tipo o modelo en concreto de periférico, dependiendo de sus características físicas y/o funcionalidades (impresoras, cámaras, tarjetas de vídeo o sonido, etc.) y típicamente desarrollados directamente por los fabricantes de los mismos.

Existen varias versiones dependiendo del sistema operativo que los emplee debido a que suelen tratar los periféricos de maneras distintas. En Linux por ejemplo, todos los dispositivos independientemente de su

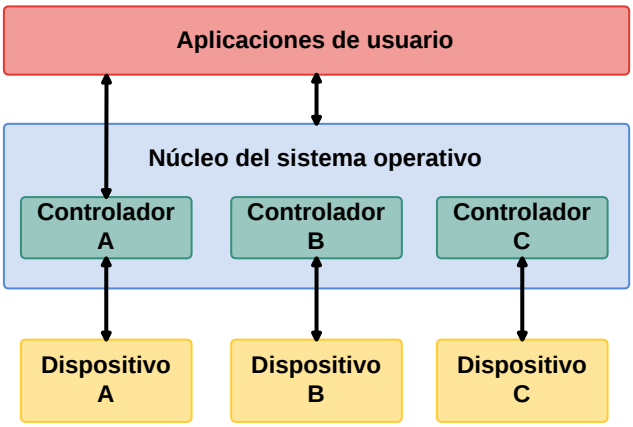


**Figura 7.1:** Esquema de organización de dispositivos y drivers

tipo son implementados como ficheros, y tras esta abstracción, está el driver interactuando directamente con el dispositivo.

## 7.3. Tipos de driver

Existen tres tipos de driver según su método de comunicación con el periférico al que sirven. En las subsecciones siguientes se procede a explicar brevemente sus tipos y un esquema de cada uno con su comportamiento.

### 7.3.1. Dispositivos de caracteres

Son aquellos a los que se accede a su contenido como un flujo de Bytes. Su driver típicamente implementará los métodos básicos para abrir y cerrar el dispositivo, y leer y escribir información. Un ejemplo de estos dispositivos son las terminales de Linux donde se ejecutan los comandos y los puertos serie antiguamente empleados por impresoras.

La diferencia más relevante entre un dispositivo de caracteres y un fichero común es que en el fichero se puede avanzar y retroceder sobre su contenido, mientas que los dispositivos sólo ofrecen un canal de entrada o salida de datos al que se accede de forma secuencial.

En la figura E.1 del anexo E, se muestra que posición ocupa este tipo de controlador integrado en el sistema, y que interfaces intervienen en su funcionamiento.

### 7.3.2. Dispositivos de bloques

Estos dispositivos son accedidos por el sistema de ficheros. En la mayoría de sistemas basados en Unix solo se pueden ejecutar instrucciones de entrada o salida del sistema al periférico, en las cuales se transmiten bloques enteros de información típicamente de 512 Bytes de longitud (o en potencias de dos más grandes).

Se diferencia respecto a un driver de caracteres en que tienen una interfaz de uso y un procesamiento de datos por parte del kernel completamente distinto.

Como se puede observar en la figura E.2, el driver recibe las páginas *cacheadas* por el sistema a través de una cola de peticiones ya implementada por el núcleo. Es el driver el encargado de instanciar su propia cola de

peticiones donde recibirá las órdenes requeridas por el *host* .

### 7.3.3. Dispositivos de red

Un controlador de un dispositivo de red es el que permite la comunicación entre los distintos *host* a través de una interfaz. Típicamente cada interfaz tiene un *hardware* asociado, pero también puede sustituirse por un *software* que virtualice la interfaz física, como por ejemplo la interfaz de *loopback*.

Los datos enviados por estos dispositivos siguen una pila de protocolos de comunicación variable según el medio y la finalidad de sus datos. Esto permite que sean muy versátiles, ya que si surge una nueva necesidad para la transmisión de datos, basta con definir un protocolo de red que se ajuste a los requisitos a cubrir e integrarlo en la pila de protocolos adjunto al mensaje. En la figura E.3 se muestra como el sistema Unix implementa este tipo de comunicación.

## 7.4. Driver `SecSys`

El módulo `SecSys` se ha desarrollado como un driver de bloques para el control de dispositivos de almacenamiento masivo con conexión USB 2.0, aportando una capa de seguridad sobre las comunicaciones y restringiendo el montaje de los dispositivos que fallen en la autenticación de credenciales. Su objetivo principal es tanto securizar el PenDrive como proteger el *host* de dispositivos no seguros impidiendo toda comunicación.

Se procede a explicar su desarrollo cronológicamente y detallando las tecnologías que han intervenido en cada una de las fases.

### 7.4.1. Creación e integración con el núcleo

El primer paso es crear un módulo para el kernel sencillo que sólo atienda a los dispositivos deseados, los cuales serán predefinidos por el programador, e imprima un mensaje en el log del núcleo cada vez que atienda un dispositivo conocido.

#### Definición de un módulo

Todo módulo posee dos funciones básicas que definen cómo se ejecuta y cómo se finaliza. En este trabajo se han definido con los prototipos mostrados en el código 7.1.

```
//Driver inicialization
static int __init usb_secsys_driver_init(void) {
    ...
}

//Driver finalization
static void __exit usb_secsys_driver_end(void) {
    ...
}

module_init(usb_secsys_driver_init);
module_exit(usb_secsys_driver_end);
```

**Código 7.1:** Funciones de inicio y finalización del módulo `SecSys`.

### Definición del módulo como driver USB

El próximo paso a cumplir es definir la utilidad del módulo que se está creando como un controlador de USB para que cumpla su función. Para ello se debe completar la estructura `usb_driver` como se muestra en el código 7.2

```
//Driver definition struct
static struct usb_driver secsys_driver = {
    .name = DRIVER_NAME,
    .id_table = secsys_table,
    .probe = usb_secsys_register_device,
    .disconnect = usb_secsys_unregister_device,
};
```

**Código 7.2:** Definición del módulo `SecSys` como driver de USB.

### Integración junto al módulo `usb-storage`

Para conseguir que el módulo sólo atienda los dispositivos deseados se debe especificar en una estructura, como se muestra en el código 7.3, indicando el Vendor y el Product del PenDrive. Pero esto no es suficiente ya que existe otro módulo USB que acepta todos los PenDrive. Esto implica que ese módulo debe ser eliminado antes de insertar el `SecSys` para que figure como primera opción en la lista de drivers USB.

```
//Driver associated device Vendor and Product
static struct usb_device_id secsys_table[ ] = {
    {USB_DEVICE(USB_SS_VENDOR_ID, USB_SS_PRODUCT_ID)},
    {}
};
```

**Código 7.3:** Definición de dispositivos conocidos por el driver `SecSys`.

## 7.4.2. Registro del dispositivo en el sistema

Para que el sistema detecte el dispositivo como una unidad de disco, el driver debe seguir unos pasos concretos para el montaje del PenDrive. Se adjunta el código 7.4 para el montaje del dispositivo y la explicación del proceso.

Primero se obtiene la información básica del dispositivo USB con la función `usb_get_dev`. Con estos datos ya se puede registrar un nuevo dispositivo de bloques con la función `register_blkdev`, la cual devuelve el número major que automáticamente se asigna por el sistema. Con esto el núcleo ya es consciente de que el periférico es un dispositivo de bloques gestionado únicamente por el driver `SecSys`.

Para permitir la llegada de órdenes del sistema se precisa de una cola de mensajes de bloques donde llegarán las peticiones que realice el sistema sobre el PenDrive que se está manejando. Es misión del driver extraer e interpretar las órdenes del núcleo recibidas en esta cola y reconstruirlas como comandos UFI (Uniform Floppy Interface.), explicados en el apartado 7.5, para la interacción con el dispositivo, y devolver el resultado de estas operaciones. Esto se debe a que el sistema no sabe cómo transmitir información directamente al dispositivo y precisa de estos drivers para realizarla. Para la creación de esta cola se emplea la función `blk_init_queue`.

Como punto final para este proceso, el módulo debe solicitar, rellenar y devolver una estructura de datos `gendisk`, para terminar de proporcionarle la información que necesita. Esta estructura se solicita ya instanciada

a través de la función `alloc_disk`. Tras definir los campos con las operaciones del driver que atenderán al dispositivo, la cola de mensajes del sistema, el número major generado por la función `register_blkdev`, el número minor y el nombre con el que se creará el disco en la partición `/dev`, se completa el proceso con la función `add_disk`. Por fin el disco ya es accesible por el sistema y puede operarse sobre él, a falta de terminar el resto del módulo.

```
static int usb_secsys_register_device(struct usb_interface* interface, const struct
    usb_device_id* id) {
...
//Get usb propieties
if ((ssd.udev = usb_get_dev(interface_to_usbdev(interface))) == NULL) {...}

//block device register
if ((ssd.major = register_blkdev(MAJOR_DEVICE, NAME_DEVICE)) < 0) {...}

//Allocate a request_queue associated with this device
if (!(secsys_blkdev_queue = blk_init_queue(secsys_blk_device_request, &secsysblkdev_lock)))
    {...}

//Register
if (!(ssd.gd = alloc_disk(NUM_MINORS))) return -EIO;
//Set the capacity of the storage media in terms of number of sectors
set_capacity(ssd.gd, secsys_blkdisk_size);
//Fill in parameters associated with the gendisk
ssd.gd->fops = &ssd.fops;
ssd.gd->queue = secsys_blkdev_queue;
ssd.gd->major = ssd.major;
ssd.gd->first_minor = 0;
sprintf(ssd.gd->disk_name, NAME_DEVICE);
add_disk(ssd.gd);
}
```

**Código 7.4:** Registro del dispositovo USB por el módulo `SecSys`.

### 7.4.3. Comunicación

La comunicación entre el driver `SecSys` y sus dispositivos sólo puede ser mediante el intercambio de mensajes conocidos como comandos UFI explicados en la sección 7.5 y a través de tuberías. A continuación se detallan los métodos empleados en el driver `SecSys` para la comunicación y todos los conceptos que intervienen en ella.

#### EndPoints

Un EndPoint es una dirección de memoria que define un punto donde el dispositivo tiene establecido un canal de comunicación con el sistema. Existen varios tipos dependiendo de su función, y cada uno de estos se encuentra en una dirección distinta para que el dispositivo identifique la clase del canal de información que se emplea. De por sí esta dirección no permite ninguna comunicación, pero sí sirven para crear las tuberías, descritas en la sección 7.4.3, que son las que realmente implementan la comunicación. Los posibles tipos de EndPoints según su función son:

- **Control:** Para el envío de comandos que realizan operaciones de control sobre el dispositivo.
- **Interrupción:** Para comunicaciones con baja cantidad de datos pero con una garantía de una transmisión constante.

- **Bulk:** Para la transmisión de datos de forma masiva.
- **Isócrono:** Para envío de datos de forma masiva pero sólo para dispositivos que admitan tolerancia a pérdidas de información.

### Tuberías

Estas tuberías son muy similares a las de un sistema operativo Linux. Se necesitan dos de ellas para una comunicación bidireccional, una de envío de mensajes y otra para su recepción. Sin embargo, a este nivel se distinguen dos tipos de tuberías para el desarrollo del módulo: tuberías de control y de Bulks.

Su construcción es sencilla pero pueden sufriese dificultades por problemas como: definir el tipo de tubería incorrecta, no usar la dirección de EndPoint adecuada, error en los parámetros de envío o recepción y roturas de las mismas por dar un uso inadecuado como por ejemplo el envío de un comando mal formado o de tipo distinto al de la tubería. Se muestra un ejemplo de uso de tubería de control y una de Bulk en el código 7.5.

```
retval = usb_control_msg(ssd.udev, usb_rcvctrlpipe(ssd.udev, ssd.bulk_in_endpointAddr -
    1), 0xfe, 0xA1, 0x0000, 0x0000, buffer_out, 0x0001, TIMEOUT_USB_CMD);

retval = usb_bulk_msg(ssd.udev, usb_sndbulkpipe(ssd.udev, ssd.bulk_out_endpointAddr),
    buffer_inquiry, BUFFER_USB_CMD_SIZE, &leido, TIMEOUT_USB_CMD);

retval = usb_bulk_msg(ssd.udev, usb_rcvbulkpipe(ssd.udev, ssd.bulk_in_endpointAddr),
    buffer_out, BUFFER_TAM, &leido, TIMEOUT_USB_CMD);
```

**Código 7.5:** Código de ejemplo para el uso de tuberías en un módulo USB.

La dirección de la tubería y del EndPoint correspondiente, siempre se considera como que el *host* es el punto de origen, es decir, las tuberías de recepción reciben datos desde el dispositivo hacia el *host* empleando el **EndPoint de entrada**, y las tuberías de envío emplean el **EndPoint de salida**.

## 7.4.4. Lectura del Superbloque y de la Tabla de particiones

Una vez aportado al sistema operativo todo lo necesario para la creación de descriptor del dispositivo, se procede a la lectura del superbloque, como muestra el código 7.6, que contiene la información básica sobre el estado de la memoria del PenDrive y la situación de la tabla de particiones.

```
//Read 10 LBA 0x00000000
if(secsys_send_cmd(buffer) > 0){
    retval = usb_bulk_msg(ssd.udev, usb_rcvbulkpipe(ssd.udev, ssd.bulk_in_endpointAddr),
        buffer_out, 4096, &leido, TIMEOUT_USB_CMD);
    retval = usb_bulk_msg(ssd.udev, usb_rcvbulkpipe(ssd.udev, ssd.bulk_in_endpointAddr),
        buffer_ack, BUFFER_TAM, &leido, TIMEOUT_USB_CMD);
}
```

**Código 7.6:** Codigo para la lectura del superbloque.

El superbloque contiene información relevante como el tamaño máximo de fichero que puede almacenar su memoria, semáforos para el control de sección crítica o la dirección donde está almacenada la tabla de particiones. Recuperando este dato ya se puede hacer otra operación de lectura para ver el particionado del dispositivo y poder interpretar correctamente los datos que contenga el PenDrive. En el caso del dispositivo empleado está en la dirección `0x0802`. Esta información que se acaba de explicar será la que primero solicite

el sistema sobre el dispositivo ya que la siguiente acción es montar en un directorio el sistema de ficheros y los datos que contenga la memoria del PenDrive. Se muestra la lectura de la tabla de particiones en el código 7.7.

```
//Read 10 LBA 0x00000802
if(secsys_send_cmd(buffer) > 0){
    retval = usb_bulk_msg(ssd.udev, usb_rcvbulkpipe(ssd.udev, ssd.bulk_in_endpointAddr),
        buffer_out, 4096, &leido, TIMEOUT_USB_CMD);
    retval = usb_bulk_msg(ssd.udev, usb_rcvbulkpipe(ssd.udev, ssd.bulk_in_endpointAddr),
        buffer_ack, BUFFER_TAM, &leido, TIMEOUT_USB_CMD);
}
```

**Código 7.7:** Código para la lectura de la tabla de particiones.

### 7.4.5. Compilación del driver

Debido a que es un módulo integrable al kernel y no un programa común, la forma de compilar cambia considerablemente. En vez de llamar al compilador desde una terminal, se debe crear un fichero Makefile que contenta una llamada al Makefile del núcleo de tal manera que, definiendo los objetos a compilar y enviándole el directorio donde está el código, construya el fichero `SecSys.ko`. La extensión de este fichero corresponde a las siglas `Kernel Object`, y es el que debe instalarse con el comando `insmod`. Se adjunta el contenido del Makefile empleado para este trabajo en el cuadro 7.5.

```
KVERSION=$(shell uname -r)
obj-m += secsys.o files.o
secsys-objs := SecSys.o files.o
all:
@make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

insert_only:
@insmod $(LKM)
```

**Cuadro 7.5:** Código del fichero Makefile para la compilación del driver `SecSys`.

## 7.5. Comandos UFI

Los comandos UFI son comandos de bloques enviados por el *host* al dispositivo UFI, que para este trabajo es el PenDrive. Cada uno tiene exactamente 12 Bytes de longitud y se identifican por el valor de su primer Byte. El formato de estos comandos viene definido en los estándares SFF-8070i [11] y en el SCSI-2 [12].

Algunos de estos comandos de bloques requieren parámetros extra o datos desde la CPU. Estos se envían al dispositivo UFI en el Bulk-EndPoint-Out como se especifica en el protocolo de transporte. Los datos que envíe el dispositivo hacia el *host* se recibirán a través del Bulk-EndPoint-In (sección 7.4.3).

### 7.5.1. Descripción de los comandos de bloques UFI

En la tabla7.1 se muestra un resumen de todos los posibles comandos de la especificación UFI junto con su código de operación y su función [13]. En la tabla 7.2 se presenta un ejemplo de la estructura interna básica de un comando UFI para proceder a continuación a definir sus campos para ayudar a comprender la formación de estos comandos, los cuales han permitido la comunicación con el dispositivo.

| Comando | Descripción | Código de operación |
|---|---|---|
| Format Unit | Formatea la unidad. | 0x04 |
| Inquiry | Obtiene la información básica del dispositivo. | 0x12 |
| Start / Stop | Pide al medio extraíble que se cargue o descargue. | 0x1B |
| Mode Select | Permite al host establecer parámetros en el dispositivo. Mode Sense debería ser prioritario al Mode Select. | 0x55 |
| Mode Select | Envía parámetros al *host*. Se requiere compatibilidad con versiones anteriores de las unidades de disquete requiere soporte para el comando Mode Sense. | 0x5A |
| Prevent/Allow Medium Removal | Impedir o permitir la retirada de los medios de comunicación a partir de una dispositivo de medios extraíbles. | 0x1E |
| Read (10) | Transfiere datos binarios desde el dispositivo al *host*. | 0x28 |
| Read (12) | Transfiere datos binarios desde el dispositivo al *host*. | 0xA8 |
| Read Capacity | Devuelve las capacidades actuales del dispositivo. | 0x25 |
| Read Format Capacity | Devuelve las capacidades actuales y los formatos soportados por el dispositivo. | 0x23 |
| Request Sense | Transferir datos de detección de estado para el anfitrión. | 0x03 |
| Rezero Unit | Sitúa la cabeza lectora del dispositivo en la posición cero. | 0x01 |
| Seek (10) | Coloca la cabeza lectora del dispositivo en la posición de memoria dada. | 0x2B |
| Send Diagnostic | Realiza un *hard reset* y ejecuta el diagnostico sobre el dispositivo. | 0x1D |
| Test Unit Ready | Solicita al dispositivo que informe sobre si está listo. | 0x00 |
| Verify | Verifica el estado de los datos en el dispositivo. | 0x2F |
| Write (10) | Transfiere datos binarios desde el *host* al dispositivo. | 0x2A |
| Write (12) | Transfiere datos binarios desde el *host* al dispositivo. | 0xAA |
| Write and Verify | Transfiere datos binarios desde el *host* al dispositivo y comprueba la integridad de los datos. | 0x2E |

**Tabla 7.1:** Tabla de comandos de bloques UFI posibles para la comunicación con dispositivos de almacenamiento USB.

A continuación se presentan los parámetros que se manejan en estos comandos y su finalidad.

#### OP code

En el primer Byte se especifica el código identificativo del comando que se ejecutará. Los posibles códigos se encuentran en la tabla 7.1 y son únicos para cada comando. La estructura de los 11 Bytes restantes dependerá del comando elegido.

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | Reserved | | | | EVPD(0) |
| 2 | Page Code | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Allocation Lenght | | | | | | | |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | Reserved | | | | | | | |
| 8 | Reserved | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla 7.2:** Ejemplo de la estructura básica para formar cualquier comando UFI. Los campos marcados como Reserved pueden rellenarse con ceros.

### LUN

El siguiente campo a especificar en cualquier comando es el LUN (Logical Unit Number), el cual comienza en el Byte 1 y ocupa los tres bits más significativos de ese Byte. Se debe obtener con el comando `GET MAX LUN`. Este valor está acotado entre 0 y 7 por limitaciones de protocolo. Es de extrema importancia guardar bien qué LUN tiene asignado cada dispositivo ya que será su identificación a la hora de enviar o recibir cualquier dato o comando. En caso de que este campo sea erróneo el sistema puede quedar gravemente afectado. Debido a sus limitaciones estará obsoleto en versiones posteriores como, por ejemplo, para el protocolo USB 3.0 se ha podido comprobar que este campo ya no se usa.

### LBA

Desde el Byte 2 al 5 encontramos un la LBA (Logical Block Address) pero no necesario en todos los comandos. Este campo sirve para direccionar sobre qué parte de la memoria del dispositivo tendrá efecto la instrucción. Su valor se calcula en función de la fórmula 7.1 [13].

$$LBA = (((trackxheadtrk) + head) \times sectrk) + (sector - 1) \tag{7.1}$$

### Transfer

La longitud de transferencia indica el volumen de los datos que se van a transferir, normalmente en numero de bloques. Para varios de los comandos este tamaño representa el número de Bytes que serán enviados como se define en la descripción de los mismos. Se debe estudiar detenidamente cada comando por si hubiera variaciones en el nombre o el uso de esta variable.

En los comandos de bloques que usen varios Bytes para este campo podemos distinguir dos casos, si el valor es cero indica que no se transferirán datos, y si el mayor o igual que 1 expresará el número de bloques que se van a transferir.

### Parameter List Length

Este parámetro también especifica el numero de Bytes que serán enviados al dispositivo UFI, pero este campo es típico de los comandos de diagnostico o cambio de parámetros del dispositivo. En caso de que se envíe un comando con este valor a 0 no es considerado error.

### Allocation Length

En este caso de especifica el numero máximo de Bytes que se reservaran en el *host* para los datos devueltos. En caso de que sea 0 no se enviarán datos pero no se considerará error. Si el valor es distinto de cero el dispositivo UFI debería terminar la transmisión de datos cuando finalice dicha operación o cuando se alcance el tamaño máximo especificado.

## 7.6. Cómo capturar tráfico USB con Wireshark

Wireshark es un *sniffer* normalmente aplicado al tráfico de una interfaz de red, pero con unos pequeños cambios puede usarse para escanear también el tráfico de un puerto USB y gracias al gran número de protocolos que tiene definidos, se puede estudiar muy cómodamente el contenido de cada paquete.

Para empezar, hay que localizar el módulo `usbmon` e insertarlo al núcleo en caso de que no esté presente, como se muestra en el código 7.8. Por defecto se puede encontrar en la ruta del kernel, en la carpeta de drivers.

```
insmod /lib/modules/3.16.0-4-amd64/kernel/drivers/usb/mon/usbmon.ko
```

**Código 7.8:** Código para insertar el módulo necesario para el uso de Wireshark como sniffer de puertos USB.

Una vez hecho esto, se puede ejecutar Wireshark siempre con permisos de usuario necesarios para tener acceso a las interfaces. Dependiendo de las características físicas del *host* donde se trabaje pueden aparecer más o menos interfaces. Se recomienda desconectar todos los medios USB para evitar confusiones en el tráfico capturado. Las interfaces de USB vendrán nombradas por el nombre del driver `usbmon` y un número identificativo.

Una vez iniciada la captura es normal que se registren unas decenas de paquetes nada más empezar sin haber insertado el dispositivo. Son parte del protocolo USB que se envía al ejecutar Wireshark sobre esa interfaz.

Debido a la escasez y dispersión de la información, este procedimiento ha sido de gran utilidad para comprender los protocolos USB y SCSI, así como funcionamiento del módulo `usb-storage` para la obtención de datos del dispositivo y para la comprobación del correcto funcionamiento del módulo de este trabajo. Ha sido necesario implementar una metodología de ingeniería inversa para obtener la información necesaria para llevar a cabo este proyecto.

# 8

# Pruebas y evaluación de resultados

Para comprobar el resultado de las fases de análisis, diseño y desarrollo se procedió a establecer una batería de pruebas para comprobar el buen funcionamiento del sistema `SecSys`. En este capítulo se exponen las pruebas realizadas y la estrategia seguida para su realización.

## 8.1. Pruebas realizadas

Las pruebas realizadas se han divido en las categorías de funcionalidad, compatibilidad, accesibilidad y usabilidad. A continuación se presentan las pruebas más significativas.

### 8.1.1. Funcionalidad

Estas pruebas están destinadas a verificar el buen funcionamiento del sistema y encontrar posibles fallos en cualquiera de los tres elementos del proyecto:

- **Prueba F1:** Se quiere comprobar la comunicación por protocolo USB entre el *host* y el dispositivo. Se detecta que se interrumpe la comunicación porque quedó incompleto el desarrollo debido a la gran complejidad del mismo, como se expondrá en el apartado de trabajo futuro.
- **Prueba F2:** El objetivo de esta prueba es verificar el procesamiento de las peticiones del sistema por el driver. Se consigue ejecutar todas las peticiones que se han recibido por el sistema pero al responderlas escribiendo su resultado no se aprecia que el sistema acepte la información devuelta. Se toma nota para finalizar correctamente esta comunicación como trabajo futuro.
- **Prueba F3:** El propósito es comprobar el correcto procesamiento de la información devuelta por el dispositivo. En su ejecución se han provocado múltiples `Kernel Panics!` por el uso de la instrucción `memcpy` en espacio de núcleo por usar estructuras de datos compactas como destino. La solución fue extraer uno a uno los datos devueltos.
- **Prueba F4:** A raíz de las pruebas realizadas anteriormente, se detectó que la lectura del fichero `/etc/passwd` en el `SecSys Authenticator` es problemático porque inducía al programa a pedir la contraseña a un usuario sin permisos de superusuario en el sistema. Simplemente se eliminó las comprobaciones sobre este fichero, lo que no afecta a su buen funcionamiento.

### 8.1.2. Compatibilidad

- **Prueba C1:** Se probó el sistema `SecSys` en otros sistemas operativos (CentOS 7 y Debian 8) y se pudo comprobar el correcto funcionamiento de todo el trabajo.
- **Prueba C2:** Dentro de este apartado, se probó el driver en otros núcleos, concluyendo que sólo funciona en versiones posteriores a la 2.6 por cambios en la estructura interna del núcleo.
- **Prueba C3:** Se intentó instalar el módulo `SecSys` en un núcleo estrictamente monolítico y como se preveía, el sistema no admite cambios en su kernel si no se recompila por completo incorporando el driver desde el principio.

### 8.1.3. Seguridad

- **Prueba S1:** Se comprobó con satisfacción que ningún usuario sin permisos no puede alterar la configuración de ninguna de las partes del sistema, ni siquiera intentar autenticarse en el sistema por el `SecSys Authenticator`.
- **Prueba S2:** De forma satisfactoria se probó que un usuario sin privilegios no puede alterar el driver `SecSys` ni reconfigurarlo. Esto es especialmente útil para evitar que usuarios sin el nivel suficiente modifiquen el funcionamiento del sistema de seguridad `SecSys`.

### 8.1.4. Usabilidad

Para estas pruebas se pidió a dos usuarios, con conocimientos informáticos equiparables a los del desarrollador, ajenos al trabajo que emplearan el sistema `SecSys` y comentaran sus dificultades y sugerencias. Es importante destacar que la usabilidad no es un objetivo prioritario para este trabajo debido a su naturaleza. Por ello se exponen a continuación las opiniones más significativas expresadas por los usuarios.

Los usuarios observaron el complejo proceso que conlleva iniciar el sistema. Esto se considera aceptable dado que, en esta fase, no está dirigido a usuarios finales.

Como sugerencia para la mejora de la usabilidad, los usuarios propusieron la implementación de un programa gráfico que manejara todas las opciones de configuración e integrara el software `SecSys Authenticator`. Además sugirieron disponer de una herramienta que, aprovechando la comunicación con el núcleo, indicará al driver que protegiera el dispositivo sin necesidad de tantísimos pasos que sólo sabría realizar el programador.

## 8.2. Evaluación de resultados

En general, los mayores fallos encontrados en el sistema `SecSys` son aspectos de usabilidad, y de falta de ciertas funcionalidades. A pesar de cumplir con los objetivos principales de autenticación y manejo de dispositivos, falta la terminación del uso del protocolo USB para dispositivos de almacenamiento tanto a nivel interno del sistema operativo como de comunicación directa con el dispositivo. Y por último, la creación de herramientas de ayuda a la usabilidad sin necesidad de alterar el código y realizar muchos pasos para operar con el dispositivo.

Respecto a su funcionamiento, tanto el driver `SecSys` cómo el programa `SecSys Authenticator`, cumplen su función de forma correcta y como se pretendía desde la definición del proyecto.

# 9

# Conclusiones y trabajo futuro

Para finalizar el trabajo se expone en el siguiente capítulo las conclusiones técnicas y personales del alumno desarrollador del mismo y un segundo apartado donde se explica el trabajo futuro que se le desea aplicar al trabajo para terminarlo en su totalidad y que pueda suponer un pequeño aporte por parte del alumno a la comunidad informática.

## 9.1. Conclusiones

Debido a la naturaleza de este trabajo, es importante expresar tanto las conclusiones técnicas como personas y hacerlo de forma separada. En las dos próximas secciones se exponen ambas.

### 9.1.1. Conclusiones Técnicas

En este trabajo se ha implementado, completamente desde el punto más elemental, un módulo para núcleo de Linux, que a modo de driver de bloques, conforma el punto más elemental de un sistema de protección de medios de almacenamiento USB, con el fin de ampliar el funcionamiento de los drivers actuales para esos dispositivos, incorporando una capa de seguridad y un sistema de autenticación de usuarios para mayor seguridad. Su funcionamiento a nivel de seguridad ha cumplido los objetivos establecidos con éxito, salvo la gestión de parte las órdenes y procesamientos asociados al protocolo USB en el que se basa su comunicación, como se explica en mayor profundidad en la sección de trabajo futuro 9.2 de este mismo capítulo.

Por otro lado se ha desarrollado un programa, llamado `SecSys Authenticator`, que empleando una interfaz gráfica sencilla y procesando ficheros del sistema solo accesibles a nivel de administrador, es capaz de determinar si las credenciales del usuario que use la aplicación son correctas para el sistema operativo. Esta decisión es comunicada al driver descrito anteriormente si debe o no continuar con el montaje del dispositivo para que sea accesible al sistema.

Aunque la gestión del dispositivo, la comunicación con él, y su montaje en el sistema son plenamente funcionales, sólo falta por concluir las respuestas desde el driver a las peticiones del sistema. Esto se debe a la complejidad del protocolo y su desarrollo, y sobretodo a la falta de información útil y la dispersión entre infinidad de libros y especificaciones de la misma.

### 9.1.2. Conclusiones Personales

Desde el principio de la realización del trabajo se sabía de la complejidad que tiene el desarrollo de partes del sistema a bajo nivel y por ello el trabajo se ha centrado al desarrollo del driver `SecSys`. Ha sido realmente

complicado encontrar información sobre el desarrollo de módulos de este tipo por ser conocimiento poco habitual al ser de tan bajo nivel y muy disperso entre multitud de documentos, libros y foros. Incluso los libros que tratan sobre el tema, como es el caso del Linux Device Drivers [14], contienen información ya obsoleta incluso en su última edición. Por ello también se tomó la decisión de ponerse en contacto con el actual desarrollador de drivers de USB, Matthew Dharm, vía e-mail para ver si podía solucionar dudas en cuanto al diseño y la comunicación con el dispositivo, a los que respondió de forma rápida y clara.

A pesar de todas las enormes dificultades y el gran ejercicio de resiliencia y la complejidad del proceso de ingeniería inversa que ha supuesto este trabajo, estoy muy contento con el tema elegido, con los resultados obtenidos a pesar de no haber podido completar el driver en su totalidad, y por la enorme cantidad de conocimientos adquiridos gracias al estudio en profundidad del sistema necesario para el desarrollo de todos los elementos del sistema `SecSys`.

También ha supuesto un reto de organización por haberse desarrollado en paralelo con prácticas laborales y con el estudio de las asignaturas del curso académico. Otro enorme logro ha sido la mejora sobre cómo, dónde y qué información se debe buscar cuando se quiere resolver dudas. Puede parecer algo trivial, pero saber localizar la información deseada y con qué términos buscarla facilita tanto el estudio de cualquier tecnología como la solución de dudas durante el desarrollo.

Como comentario final me gustaría añadir que a pesar de las etapas más complicadas y duras de este proyecto, estoy satisfecho con el resultado y me gustaría terminarlo como proyecto personal para, con suerte, publicarlo en algún repositorio público de Linux.

## 9.2. Trabajo futuro

Las líneas futuras de desarrollo son básicamente las tres que se exponen a continuación.

Primero, y el más importante, terminar toda la comunicación y el correcto procesamiento de todos el protocolo USB, ya que ha sido uno de los puntos quizá más ambiciosos y no se ha podido completar en su totalidad. Esto requiere de un conocimiento muy profundo y detallado de muchas partes del sistema, como por ejemplo, unas estructuras de datos que indican direcciones de memoria para escritura de datos, y otros sistemas de particionado de datos.

En segundo lugar, la mejora del programa de autenticación `SecSys Authenticator`. A pesar de que cumpla su función elemental, su método de comunicación con el núcleo es muy rudimentario, y se debería mejorar y ampliar la funcionalidad para la ayuda de la configuración del sistema y la creación, ahora muy compleja, de un PenDrive securizado.

Y como tercer punto y no menos importante, añadir sistemas de cifrado a través de los módulos destinados para ello del propio núcleo, cifrado del superbloque y de la tabla de particiones, y establecer múltiples opciones de cifrado y distintas contraseñas para usuario y dispositivo para dificultar el robo de datos del dispositivo y que el usuario que lo emplee tenga total certeza de que su información está a salvo.

# BIBLIOGRAFÍA


[1] C. Alonso, "Thinking about Security." https://www.youtube.com/watch?v=Rxs9meo9vwQ. Accessed: 2015-03-03.

[2] "Stuxnet attackers used 4 Windows zero-day exploits." http://www.zdnet.com/article/stuxnet-attackers-used-4-windows-zero-day-exploits/, September 2014. Accessed: 2016-02-01.

[3] "7 Razones por las que Linux es más usado que Windows en servidores." http://hipertextual.com/archivo/2014/05/linux-servidores/. Accessed: 2016-06-01.

[4] W. Mauerer, *Professional Linux Kernel Architecture*. Wiley Publishing, Inc., 2008.

[5] "Linux Kernel Map." http://www.makelinux.net/kernel_map/. Accessed: 2016-02-15.

[6] J. Axelson, *USB Mass Storage: Designing and Programming Devices and Embedded Hosts*. Independent Publishers Group, 2006.

[7] L. Torvalds, "Linux Code Repository." https://github.com/torvalds/linux. Accessed: 2016-02-12.

[8] C. Saout, "http://linux.die.net/man/8/cryptsetup," tech. rep.

[9] "Rohos Mini Drive." http://es.ccm.net/faq/6845-proteger-los-datos-de-una-memoria.-usb-con-una-contrasena. Accessed: 2016-06-15.

[10] "GNU Bash." https://www.gnu.org/software/bash/, May 2016. Accessed: 2016-04-20.

[11] S. Committe, "SFF-8070i Specification for ATAPI Removable Rewritable Media Devices," tech. rep., July 2001.

[12] D. Ashby and J. Averyt, "Disc Drive SCSI-2/SCSI-3 Interface," tech. rep., August 1997.

[13] K. Hamada, M. Kubo, J. Blackson, *et al.*, "Universal Serial Bus Mass Storage Class. UFI Command Specification," Tech. Rep. 1.0, December 1998.

[14] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers*. O'Really, third ed., February 2005.

[15] N. West, "Linux System Errors." http://www-numi.fnal.gov/offline_software/srt_public_context/WebDocs/Errors/unix_system_errors.html. Accessed: 2016-06-11.

# I

# GLOSARIO

# ACRÓNIMOS

EVPD ..... Enable Vital Product Data

GPL ..... General Public License

LBA ..... Logical Block Address

LUN ..... Logical Unit Number

RMB ..... Removable Media Bit

SCSI ..... Small Computer System Interface

UFI ..... Uniform Floppy Interface

# DEFINICIONES

**0-days**

*Software malicioso que aprovecha una vulnerabilidad sin resolver.*

**AES**

*Algoritmo de cifrado.*

**Bulk**

*Tipo de EndPoint para la transmisión de datos de forma masiva.*

**credenciales**

*Combinación de nombre de usuario y contraseña*

**ciber-ataque**

*Ataque contra una infraestructura informática.*

**driver**

*Módulo del kernel para el control de periféricos.*

**firewall**

*Software destinado a la protección de un sistema informático que actúa como barrera para el tráfico de red no deseado.*

**hacker**

*Profesional de la seguridad informática dedicado a la búsqueda de vulnerabilidades en sistemas informáticos.*

**kernel**

*Núcleo del sistema operativo.*

**Linus Torvalds**

*Autor del sistema Linux.*

**log**

*Fichero donde se recopilan todos los mensajes emitidos por un programa para que puedan ser examinados por el desarrollador o por los usuarios tanto como para localizar fallos como para ver las acciones que se llevan a cabo.*

**major**

*Número que identifica a un driver en el sistema operativo.*

**minor**

*Número para la identificación de dispositivos atendidos por un driver.*

**PenDrive**

*Dispositivo de almacenamiento extraíble con conexión USB.*

**Product**

*Número de 2 Bytes que, en un dispositivo, identifica el modelo del dispositivo que ha construido un determinado fabricante.*

**script**

*Pequeño programa formado por un conjunto de instrucciones con el objetivo de automatizar tareas aprovechando la potencia que ofrece la linea de comandos. En este trabajo sólo se consideran scripts en Bash.*

**SCSI** *Small Computer System Interface.*

**sección crítica**

*Parte de la ejecución de un programa sensible a problemas de concurrencia, cuando se trabaja con varios procesos o hilos, que debe ser accedida sólo por un programa o hilo de forma simultánea para evitar inestabilidades.*

**semáforo**

*Sistema de control de concurrencia de procesos o hilos que indica si la sección crítica esta disponible de acceso o está ocupada por otro.*

**sniffer**

*Programa informático para la captura de información entre el host donde se ejecuta y sus puertos de conexión con periféricos o redes.*

**estructuras de datos compactas**

*Se dice que una estructura de datos es compacta cuando todos sus campos están alineados en el espacio de memoria.*

**superbloque**

*Bloque principal alojado en la dirección cero. Contiene la información básica sobre el estado de la memoria del dispositivo y sus características así como la referencia la tabla de particiones.*

**tabla de particiones**

*Estructura donde se definen las particiones, con su formato, tamaño y características para que pueda ser correctamente interpretada la información de un medio de almacenamiento por parte del sistema operativo.*

**Thinking about Security**

*Conferencia impartida por Chema Alonso de la cual surje la idea de este trabajo. La exposición de los hechos descritos en la motivación comienza en el minuto 58:00.*

**UFI** *Uniform Floppy Interface.*

**Unix**

*Núcleo en el que se basa Linux.*

**USB** *Universal Serial Bus.*

**Vendor**

*Número de 2 Bytes que, en un dispositivo, identifica al fabricante encargado de la construcción de ese dispositivo. Un mismo fabricante puede tener más de un número Vendor.*

**virus**

*Software malicioso.*

# A

# Versiones Kernel Linux

El versionado del núcleo sigue una estructura de 4 números, cada uno con un significado. Un ejemplo de esta numeración sería X.Y.Z.W

- **X:** Da nombre a la versión de núcleo.
- **Y:** Denota la subversión del núcleo denominado por X.
- **Z:** Usado para indicar el número de la revisión del núcleo.
- **W:** Empleado para señalar la reparación de un error grave.

| Versión | Fecha de lanzamiento |
|---|---|
| 0.01 | septiembre de 1991 |
| 0.02 | octubre de 1991 |
| 0.11 | diciembre de 1991 |
| 1.0.0 | marzo de 1994 |
| 1.2.0 | marzo de 1995 |
| 2.2.0 | enero de 1999 |
| 2.4.0 | enero de 2001 |
| 2.6.0 | diciembre de 2003 |
| 2.6.28 | diciembre de 2008 |
| 2.6.36 | octubre de 2010 |
| 3.0 | julio de 2011 |
| 3.3.6 | mayo de 2012 |
| 3.4 | mayo de 2012 |
| 3.19 | febrero de 2015 |
| 4.0 | abril de 2015 |
| 4.6 | mayo de 2016 |

**Tabla A.1:** Tabla de las versiones del kernel de Linux

# B

# Estructuras de datos del módulo SecSys

## B.1. Estructuras de datos

Estas son las estructuras de datos empleadas para el desarrollo del driver `SecSys`.

### B.1.1. Estructura de datos principal para el driver `SecSys`

```
typedef struct SecSys_Data {
    //general data
    int         new_device;     //Flag if is new device


    //USB
    struct usb_device*    udev;     //USB propieties
    u8          use;      //is in use Flag
    unsigned long long    size;     //Media size
    struct gendisk*       gd;       //gendisk struct
    u8*         data;     //data Buffer
    struct block_device_operations fops;      //Driver operations
    int         major;      //Device Major num (0 = dinamic)
    struct request_queue*   secsys_blkdev_queue; //System requests queue
    int         lun;      //Logical Unit Number

    //EndPoints
    int         bulk_in_endpointAddr; //Fisical address of the Input data EndPoint
    char*         bulk_in_buffer;   //Input Buffer
    int         bulk_in_size;   //Input Buffer size
    int         bulk_out_endpointAddr; //Fisical address of the Output data EndPoint
    char*         bulk_out_buffer;  //Output Buffer
    int         bulk_out_size;    //Output Buffer size
    int         ctrl_endpointAddr;  //Fisical address of the control EndPoint

    //Requests Data
    struct inquiryData    inqData;    //Inquiry Request Data
} secsys_data;
```

**Código B.1:** Definición de la estructura de datos del driver `SecSys` diseñada por el alumno para el manejo de todas las variables asociadas al driver durante su ejecución para el montaje del dispositivo.

### B.1.2. Estructura de datos del comando Inquiry

```
typedef struct inquiryData {
    __u8       peripheralDeviceType; //Device Tipe
    __u8       rmb;     //
    __u8       ISOVersion;    //
    __u8       ECMAVersion;   //
    __u8       ANSIVersion;   //ANSI Codification version
    __u8       ResponeDataFormat;  //
    __u8       addLength;    //Additional length
    char       vendor[9];    //Vendor Information
    char       product[17];   //Product Information
    __u32      productRevisionLevel; //
} inquiry_data;
```

**Código B.2:** Definición de la estructura de datos de los datos retornados por el comando Inquiry construida a partir del esquema de datos que devuelve el comando como resultado de su ejecución.

### B.1.3. Macros definidas para el uso del driver

```
#define ERR -1
#define OK 0
#define FALSE 0
#define TRUE !(FALSE)
```

**Código B.3:** Definición de macros con valores más elementales para las comprobaciones de éxito o fallo en las funciones del driver `SecSys`

```
#define NAME "SecSys v0.2.2"
#define MSG_HEADER "[["NAME"]]-"
#define PRINT_FC printk
#define PRINT_MSG(msg) PRINT_FC(KERN_INFO MSG_HEADER "-" msg "\n")
#define PRINT_WARN(msg) PRINT_FC(KERN_WARNING MSG_HEADER
    "-[[WARNING]] " msg "\n")
#define PRINT_ALERT(msg) PRINT_FC(KERN_ALERT MSG_HEADER "-[[ALERT]] "
    msg "\n")
#define PRINT_DATA(msg, arg) PRINT_FC(KERN_ALERT MSG_HEADER "-[[DATA]] "
    msg "\n", arg)

#define DEBUG_MODE
```

```
#ifdef DEBUG_MODE //Debug print functions
#define DEBUG KERN_DEBUG
#define PRINT_ERR(msg) PRINT_FC(KERN_ERR MSG_HEADER "-[[ERROR]] " msg "
    In line %d of function %s on file %s\n" ,__LINE__, __FUNCTION__, __FILE__)
#define PRINT_DEBUG(msg) PRINT_FC(DEBUG MSG_HEADER "-[[DEBUG]] " msg " In
    line %d of function %s on file %s\n" ,__LINE__, __FUNCTION__, __FILE__)
#define PRINT_HEX(buff,len) {int i; printk(KERN_NOTICE "ASCII "); for(i = 0; i < (len) ;
    ++i) printk(KERN_NOTICE " %d=%02x ",i,(buff)[i]); printk(KERN_NOTICE "\n");}
#else     //Normal print functions
#define DEBUG KERN_NOTICE
#define PRINT_ERR(msg) PRINT_FC(KERN_ERR MSG_HEADER "-[[ERROR]] " msg
    "\n")
#define PRINT_DEBUG(msg) PRINT_FC(DEBUG MSG_HEADER "-[[NOTICE]] " msg
    "\n")
#endif
```

**Código B.4:** Definición de macros para agilizar la escritura de mensajes en el log del kernel según su importancia empleadas por el driver `SecSys`

### B.1.4. Estructura de datos del formato Ext 2

```
struct ext2_super_block {
    __u32 s_inodes_count;   //Inodes count
    __u32 s_blocks_count;   //Blocks count
    __u32 s_r_blocks_count;  //Reserved blocks count
    __u32 s_free_blocks_count; //Free blocks count
    __u32 s_free_inodes_count; //Free inodes count
    __u32 s_first_data_block;  //First Data Block
    __u32 s_log_block_size;  //Block size
    __s32 s_log_frag_size;  //Fragment size
    __u32 s_blocks_per_group;  //# Blocks per group
    __u32 s_frags_per_group;  //# Fragments per group
    __u32 s_inodes_per_group;  //# Inodes per group
    __u32 s_mtime;    //Mount time
    __u32 s_wtime;    //Write time
    __u16 s_mnt_count;   //Mount count
    __s16 s_max_mnt_count;  //Maximal mount count
    __u16 s_magic;    //Magic signature
    __u16 s_state;    //File system state
    __u16 s_errors;    //Behaviour when detecting errors
    __u16 s_minor_rev_level;   //minor revision level
    __u32 s_lastcheck;   //time of last check
```

```
    __u32 s_checkinterval;  //max. time between checks
    __u32 s_creator_os;   //OS
    __u32 s_rev_level;   //Revision level
    __u16 s_def_resuid;   //Default uid for reserved blocks
    __u16 s_def_resgid;   //Default gid for reserved blocks
    // These fields are for EXT2_DYNAMIC_REV superblocks only.
    // Note: the difference between the compatible feature set and the incompatible
    // feature set is that if there is a bit set in the incompatible feature set
    // that the kernel doesn't know about, it should refuse to mount the filesystem.
    // e2fsck's requirements are more strict; if it doesn't know about a feature in
    // either the compatible or incompatible feature set, it must abort and not try
    // to meddle wit/things it doesn't understand...
    __u32 s_first_ino;   //First non-reserved inode
    __u16 s_inode_size;   //size of inode structure
    __u16 s_block_group_nr;  //block group # of this superblock
    __u32 s_feature_compat;  //compatible feature set
    __u32 s_feature_incompat;  //incompatible feature set
    __u32 s_feature_ro_compat;  //readonly-compatible feature set
    __u8 s_uuid[16];  //128-bit uuid for volume
    char s_volume_name[16];  //volume name
    char s_last_mounted[64];  //directory where last mounted
    __u32 s_algorithm_usage_bitmap; //For compression
    // Performance hints. Directory preallocation should only
    // happen if the EXT2_COMPAT_PREALLOC flag is on.
    __u8 s_prealloc_blocks; //Nr of blocks to try to preallocate//
    __u8 s_prealloc_dir_blocks; //Nr to preallocate for dirs //
    __u16 s_padding1;
    __u32 s_reserved[204]; //Padding to the end of the block //
};
```

**Código B.5:** Definición de la estructura de datos del formato Ext2 que utiliza el driver para la interpretación de la tabla de particiones.

# C

# Prototipos de las funciones del sistema SecSys

## C.1. Prototipos del Drvier SecSys

```
//Prototipos
static int usb_secsys_register_device(struct usb_interface* interface, const struct
    usb_device_id* id);
static void usb_secsys_unregister_device(struct usb_interface* interface);
static void printDiskInfo(struct SecSys_Data* ssd);
static void printSecSysDataInfo(void);
static int secsys_init_secsys_drive(void);
static void secsys_blk_device_request(struct request_queue* rq);
static int secsys_send_cmd(__u8 *pdu);
static int secsys_get_file_system(void);
static int secsys_transfer2(struct request *req);
static void secsys_transfer(struct SecSys_Data* dev, sector_t sector, unsigned long nsect,
    char* buffer, int write);
static int init_buffer_pdu(char* buf, int len);
static void extractInquiryData(struct SecSys_Data *, char* buf, int len);
static void printInquiryData(struct SecSys_Data *ssd);
static void printExt2Info(struct ext2_super_block *esb);
static int readCredentials(void);
static int errorCodePDU(int retval);
```

**Código C.1:** Definición de los prototipos de las funciones empleadas para el desarrollo del driver `SecSys`

```
struct file* file_open(const char* path, int flags, int rights);
void file_close(struct file* file);
int file_read(struct file* file, unsigned long long offset, unsigned char* data, unsigned int
    size);
int file_write(struct file* file, unsigned long long offset, unsigned char* data, unsigned int
    size);
int file_sync(struct file* file);
```

**Código C.2:** Definición de los prototipos de las funciones de manejo de ficheros empleadas por el driver `SecSys`

## C.2. Prototipos del `SecSys Authenticator`

```
typedef struct USER_UNIX{
    char user[USER_NAME_SIZE];
    char encryp_method[METHOD_SIZE];
    char salt[SALT_SIZE];
    char passwd[PASS_SIZE];
}USER_UNIX;

char *getSaltUser(const char *user);
char *getMethodUser(const char *user);
char *getPassUser(const char *user);
bool isUserOnSystem(const char *user);
void freeUser();
```

**Código C.3:** Definición de los prototipos de las funciones para el manejo de usuarios de Unix

```
USER_UNIX *parseShadowFile(const char *user);
```

**Código C.4:** Definición de los prototipos de las funciones para el procesamiento del fichero `/etc/shadow`

```
void setOutChannel();

void closeOutChannel();

void printInform(const int errorCode, const bool error, const int line, const char *file);

void printErrorMessage(const int errorCode, const int line, const char *file);

void printInformMessage(const int errorCode);

char *getErrorMessage(const int errorCode);


#define CODE_OK 0
#define CODE_DEFAULT_ERROR 1
#define CODE_NO_USER_PARSE_SHADOW_FILE 400
#define CODE_MALLOC_ERROR 401
#define CODE_FILE_OPEN_ERROR 402
#define CODE_SHADOW_FILE_IS_OK 200
#define CODE_SHADOW_FILE_IS_ERR 201
#define CODE_PASSWORD_CORRECT 202
#define CODE_PASSWORD_INCORRECT 203
#define CODE_PASSWD_FILE_IS_OK 172
#define CODE_PASSWD_FILE_IS_ERR 13
#define CODE_COMMAND_RELAUNCH_ERR 204
#define CODE_COMMAND_RELAUNCH_OK 205
#define CODE_USER_NOT_FOUND_ON_SYSTEM_CODE 123
#define CODE_NOT_VALID_METHOD 312
#define CODE_WRITE_CREDENTIALS_ERR 412
#define CODE_CREDENTIALS_OK 340
#define CODE_CREDENTIALS_ERR 341
#define FILE_ERR out_err
#define FILE_MSG out_msg
```

**Código C.5:** Definición de los códigos de error y de las funciones para el uso del log usados por `SecSys Authenticator`

# D

# Otros comandos USB-SCSI

En este anexo se detallan algunos de los comandos para la comunicación entre el *host* y el dispositivo más significativos utilizados en el driver `SecSys`. Toda esta información se puede encontrar más detallada en el documento [13].

## D.1. Get Max LUN

El primer paso antes de mandar cualquier PDU es encontrar el LUN, descrito en la sección 7.5.1, al que responde el dispositivo que manejará el driver, ya que se requerirá para construir los comandos de bloques. En el caso del driver `SecSys`, como está diseñado para atender sólo un dispositivo simultáneamente, no afecta gravemente a su funcionamiento porque siempre será 0. Pero si se quisiera manejar varios a la vez debe manejar cuidadosamente este valor para enviar los comandos al dispositivo adecuado. Al ser un comando de control debe ser enviado a través del EndPoint de control para que sea interpretado por el dispositivo y ejecutado correctamente. En el caso del dispositivo usado es el EndPoint residente en la dirección 0x80. Se adjunta en el código D.1 para mostrar la particularidad de su envío.

```
retval = usb_bulk_msg(ssd.udev, usb_rcvbulkpipe(ssd.udev,
    ssd.bulk_in_endpointAddr), buffer_out, 4096, &leido, TIMEOUT_USB_CMD);
PRINT_DATA("RETORNO: %d", retval);
```

**Código D.1:** Envío mensaje GET MAX LUN por el EndPoint de control.

## D.2. Inquiry

El comando Inquiry solicita la información del dispositivo UFI y es enviada al *host*. Esta información suele usarse por los drivers después de arrancar el dispositivo o de realizar un *hardware reset*. Su código de operación es 0x12. En la tabla D.1 se muestra la estructura de los campos internos del comando.

El parámetro EVPD (Enable Vital Product Data) debe inicializarse a cero.

El parámetro *Page Code* especifica en que pagina de datos sobre el dispositivo UFI debería devolver al *host*. Estos dispositivos admiten sólo el número de página 0x00, la cual contiene los datos descritos más adelante en la tabla D.2.

El parámetro de *Allocation Length* especifica cuánta información se enviará al *host* en este comando medido

en Bytes. Si este valor es cero, no causará un error, pero tampoco devolverá información. Para poder retornar todos los datos D.2 se necesitan al menos 36 Bytes.

Este comando no reporta información acerca de los cambios que hayan podido suceder en el dispositivo, o si está listo o no. A su vez, tampoco tiene efecto en la unidad o su estado.

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | Reserved | | | | EVPD(0) |
| 2 | Page Code | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Allocation Lenght | | | | | | | |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | Reserved | | | | | | | |
| 8 | Reserved | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.1:** Tabla de la estructura para formar un comando de tipo Inquiry.

## D.2.1. Inquiry Data

Los datos devueltos por el comando Inquiry a través del Bulk_EndPoint_in siguen la estructura mostrada en la tabla D.2.

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Reserved | | | Peripheral Device Type | | | | |
| 1 | RMB | Reserved | | | | | | |
| 2 | ISO Version | | ECMA Version | | | ANSI Version (00h) | | |
| 3 | Reserved | | | | Response Data Format | | | |
| 4 | Additional Length | | | | | | | |
| 5 .. 7 | Reserved | | | | | | | |
| 8 .. 15 | Vendor Information | | | | | | | |
| 16 .. 31 | Product Information | | | | | | | |
| 32 .. 35 | Product Revision Level | | | | | | | |

**Tabla D.2:** Tabla de la estructura de los datos devuelto al EndPoint al ejecutar un comando de tipo Inquiry.

El Peripheral Device Type especifica el tipo del dispositivo:

1.– **0x00:** Dispositivo de acceso directo.

2.– **0x1F:** Ninguno. No hay disco conectado en el LUN especificado.

*RMB (Removable Media Bit)*: Si está a 1 indica que el dispositivo se puede desconectar. Si su valor es 0, el dispositivo puede desconectarse.

*ISO/ECMA:* Este campo debería ser 0.

*ANSI Version:* Debería ser 0 en esta especificación.

*Response Data Format:* Debería ser 0x01 para esta especificación.

*Additional Length:* Especifica la longitud en Bytes de los parámetros devueltos. Si la longitud enviada en el comando solicitante es demasiado pequeña para transferir todos los parámetros se reflejará en este campo. El medio extraíble debería marcar este valor a 0x1F.

*Product Revision Level:* Contiene 4 Bytes con la definición del producto dada por el Vendor. Para un dispositivo UFI el campo indicará el número de la versión del *firmware*.

## D.3. Test Unit Ready

Este comando proporciona información sobre la disponibilidad del dispositivo UFI. Su retorno abarca tres posibles valores según la disponibilidad del dispositivo. El valor *GOOD* indica que el dispositivo es accesible, *CHECK-CONDITION* indica algún tipo de error que debe ser tratado, y *NOT READY* significa que el dispositivo está ocupado con otra acción en el momento de la ejecución. En la tabla D.3 se muestra la estructura interna de los valores que conforman este comando. Su código de operación es el 0x00.

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | | Reserved | | | |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Reserved | | | | | | | |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | Reserved | | | | | | | |
| 8 | Reserved | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.3:** Tabla de la estructura para formar un comando de tipo Test Unit Ready.

## D.4. Request Sense

El comando Request Sense ordena al dispositivo transferir un valor Sense Data. Estos datos referentes al comando ejecutado con anterioridad a este son devueltos en el Bulk_EndPoint_in. El *host* debería implementar esta orden tras cada comando para comprobar si ocurrió algún error. En la tabla D.4 se muestra la estructura interna del comando. Su código de operación es 0x03.

El campo *Allocation Length* especifica el numero máximo de Bytes de *SenseData* que el *host* puede recibir.

El dispositivo UFI preservará los datos hasta que se sobrescribe con el resultado de la ejecución del siguiente comando de bloques. El dispositivo no debería cambiar los datos de los *SenseData* hasta la terminación del siguiente *REQUEST-SENSE*. Los datos devueltos por esta instrucción se muestran en la tabla D.5.

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | Reserved | | | | |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Allocation Length | | | | | | | |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | Reserved | | | | | | | |
| 8 | Reserved | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.4:** Tabla de la estructura para formar un comando de tipo Request Sense.

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Valid | Error Code | | | | | | |
| 1 | Reserved | | | | | | | |
| 2 | Reserved | | | | Sense Key | | | |
| 3 | (MSB) | | | | | | | |
| 4 | Information | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | (LSB) |
| 7 | Additional Sense Length | | | | | | | |
| 8 | Reserved | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |
| 11 | | | | | | | | |
| 12 | Additional Sense Code | | | | | | | |
| 13 | Additional Sense Code Qualifier | | | | | | | |
| 14 | Reserved | | | | | | | |
| 15 | Reserved | | | | | | | |
| 16 | | | | | | | | |
| 17 | | | | | | | | |

**Tabla D.5:** Tabla de la estructura de datos de un comando de tipo Request Sense.

## D.5. Read Capacity

Este comando permite al *host* conocer las posibilidades de comunicación que ofrece el dispositivo. En la tabla D.6 se muestra la estructura interna del comando. Su código de operación es 0x25.

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | Reserved | | | | RelAdr |
| 2 | (MSB) | | | | | | | |
| 3 | LBA | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | Reserved | | | | | | | |
| 7 | Reserved | | | | | | | |
| 8 | Reserved | | | | | | | PMI |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.6:** Tabla de la estructura para formar un comando de tipo Read Capacity.

Los campos *RelAdr*, *LBA* y *PMI* deben valer 0.

Si el dispositivo UFI reconoce el medio formateado, el comando devolverá al *host* los datos descritos en la figura D.7, por el EndPoint de lectura de datos. Si el dispositivo UFI establece el código *NO SENSE* significa que, o no está formateado, está desconocido, o no está presente. En este caso establece el valor *SenseKey* a uno de los valores posibles destacados en la sección D.4.

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | | | | | |
| 1 | Last LBA | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | (LSB) |
| 4 | (MSB) | | | | | | | |
| 5 | Longitud del bloque en Bytes | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | (LSB) |

**Tabla D.7:** Tabla de la estructura de datos devuelta por un comando de tipo Read Capacity.

En los datos devueltos por este comando vemos únicamente 2 campos con un tamaño total de 8 Bytes:

El campo *Last LBA* contiene la última LBA para usarse con los comandos de acceso de datos al dispositivo.

El campo *Longitud del bloque en Bytes* especifica la longitud en Bytes de cada bloque para el descriptor de capacidades.

## D.6. Mode Sense

El comando *Mode Sense* permite al dispositivo UFI devolver parámetros del medio o del dispositivo al *host*. Es complementario al comando *MODE SELECT*. En la tabla D.8 se muestra la estructura interna del comando. Su código de operación es 0x5A.

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | Reserved | DBD | Reserved | | |
| 2 | PC | | Page Code | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Reserved | | | | | | | |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | Parameter List Length (MSB) | | | | | | | |
| 8 | Parameter List Length (LSB) | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.8:** Tabla de la estructura para formar un comando de tipo Mode Sense.

El campo *DBD* debe ser 0 para este tipo de dispositivos.

El campo *Page Code* especifica en que modo se devuelven las páginas a devolver.

## D.7. Medium Removal

Este comando comunica al dispositivo UFI que active o desactive la extracción del medio. En la tabla D.9 se muestra la estructura interna del comando. Su código de operación es 0x1E.

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | Reserved | | | | |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Reserved | | | | | | | Prevent |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | Reserved | | | | | | | |
| 8 | Reserved | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.9:** Tabla de la estructura para formar un comando de tipo Medium Removal.

El único parámetro que podemos modificar en este comando es el bit que corresponde al atributo *Prevent*,

el cual puede tener dos valores segun su funcionalidad:

1.– Activa/permite la extracción del medio.

2.– Desactiva/previene la extracción del medio.

Obviamente el dispositivo puede extraerse sin ejecutar este comando pero ello puede provocar errores en el dispositivo o corrupción de datos si no se ejecutan correctamente los comandos de control como este.

## D.8. Read-10

Este comando realiza peticiones de lectura al dispositivo UFI. Los datos serán escritos en la dirección indicada por el campo LBA. En la tabla D.10 se muestra la estructura interna del comando. Su código de operación es 0x28.

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | DPO | FUA | Reserved | | RelAdr |
| 2 | (MSB) | | | | | | | |
| 3 | LBA | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | Reserved | | | | | | | |
| 7 | Transfer Length (MSB) | | | | | | | |
| 8 | Transfer Length (LSB) | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.10:** Tabla de la estructura para formar un comando de tipo Read(10).

*DPO:* Debería valer 0 para esta especificación.

*FUA:* Debería valer 0 para esta especificación.

*RelAdr:* Debería valer 0 para esta especificación.

## D.9. Write-10

Este comando realiza peticiones de escritura al dispositivo UFI. En la tabla D.11 se muestra la estructura interna del comando. Su código de operación es 0x2A.

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation Code | | | | | | | |
| 1 | Logial Unit Number | | | DPO | FUA | Reserved | | RelAdr |
| 2 | (MSB) | | | | | | | |
| 3 | LBA | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | Reserved | | | | | | | |
| 7 | Transfer Length (MSB) | | | | | | | |
| 8 | Transfer Length (LSB) | | | | | | | |
| 9 | Reserved | | | | | | | |
| 10 | Reserved | | | | | | | |
| 11 | Reserved | | | | | | | |

**Tabla D.11:** Tabla de la estructura para formar un comando de tipo Write(10).

*DPO:* Debería valer 0 para esta especificación.

*FUA:* Debería valer 0 para esta especificación.

*RelAdr:* Debería valer 0 para esta especificación.

# E

# Esquemas de funcionamiento de drivers de Linux

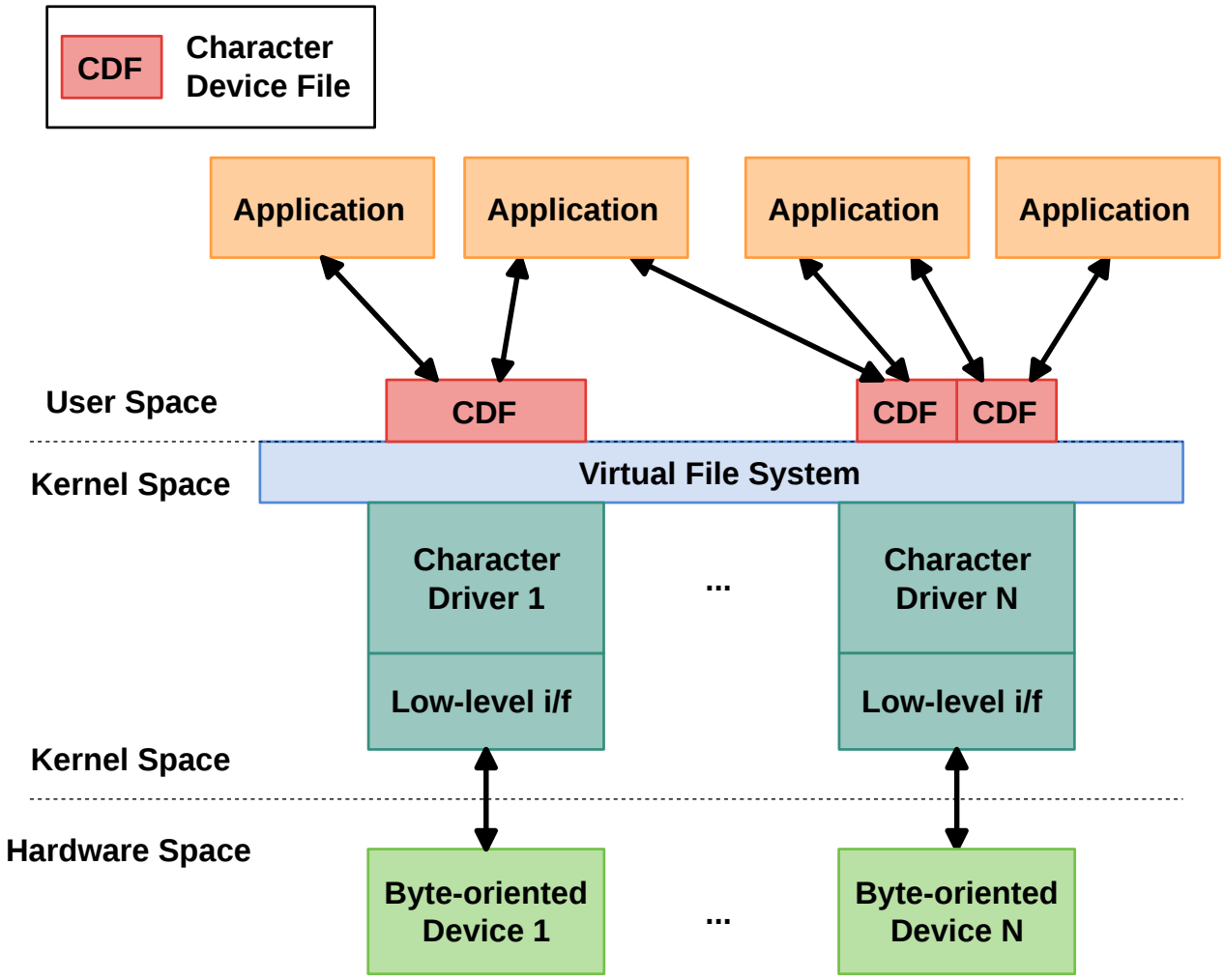


**Figura E.1:** Esquema del funcionamiento de un driver de caracteres donde se muestran sus interacciones con el dispositivo que controlan y el sistema al que sirven.

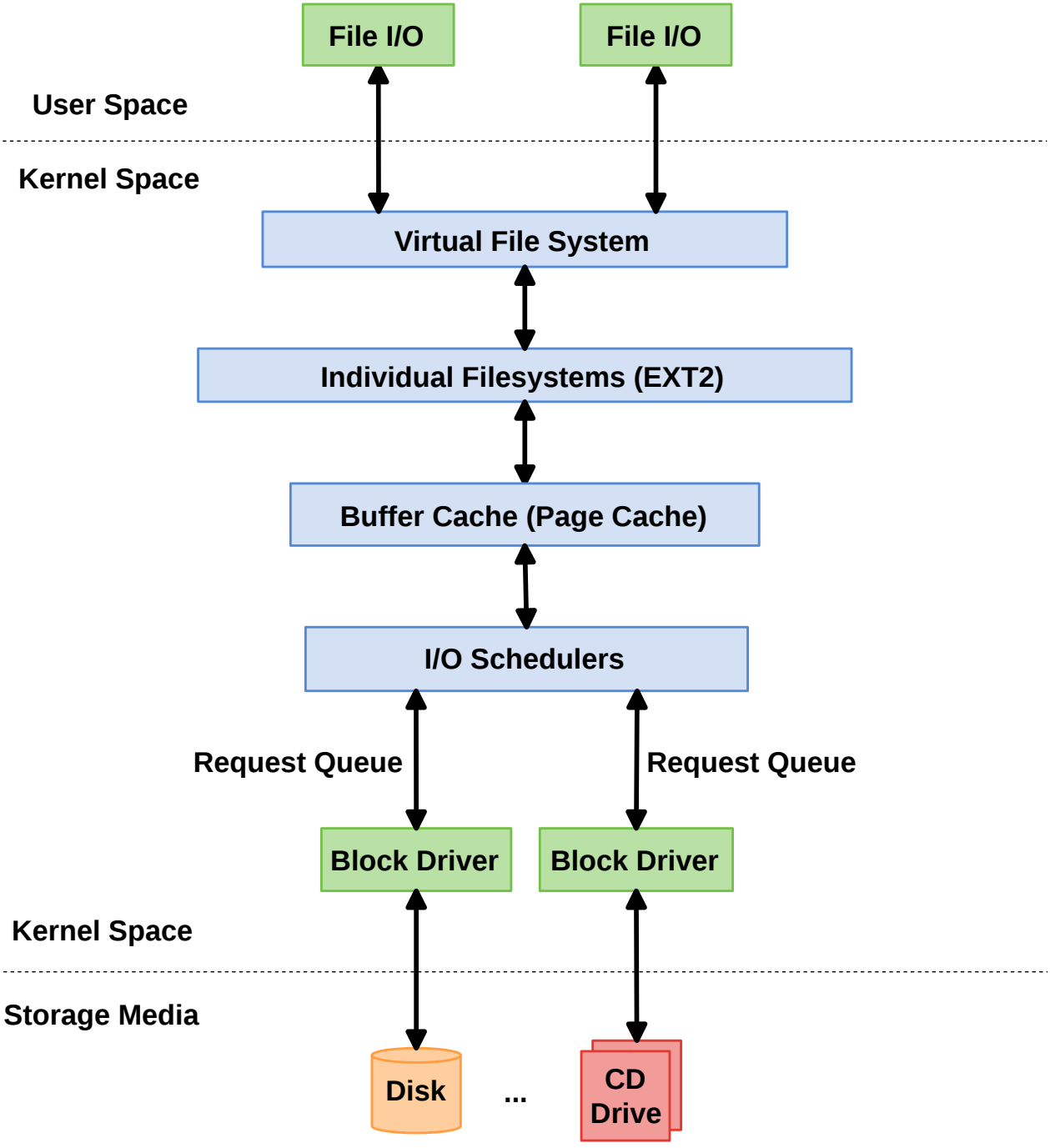


**Figura E.2:** Esquema del funcionamiento de un driver de bloques donde se muestran sus interacciones con el dispositivo que controlan y el sistema al que sirven.

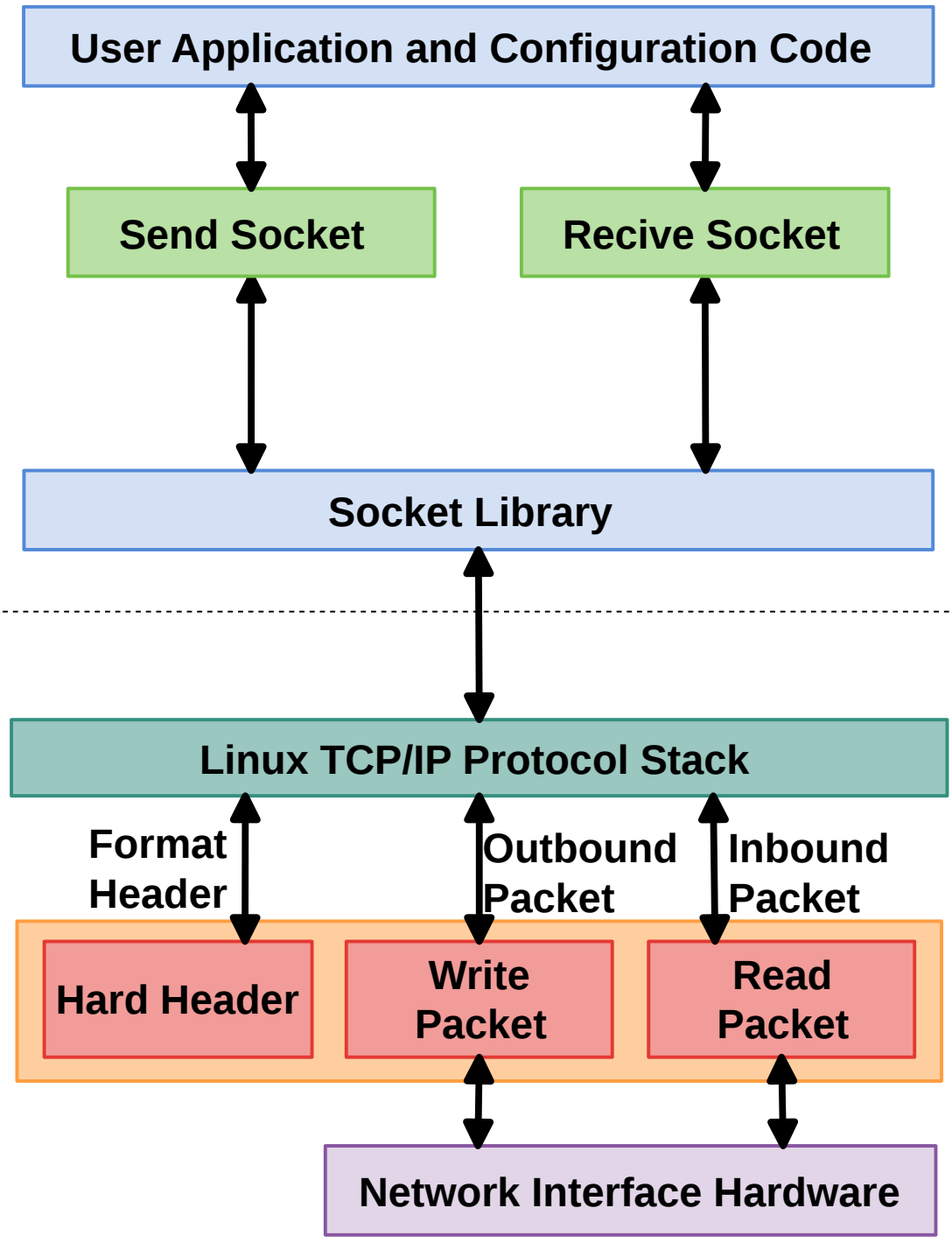


**Figura E.3:** Esquema del funcionamiento de un driver de red donde se muestran sus interacciones con el dispositivo que controlan y el sistema al que sirven.

# F

# Códigos de error posibles en el núcleo

En este apéndice se muestran todos los códigos de error documentados para el uso del núcleo de Linux. A pesar de que sus valores sean valores positivos, se deben emplear siempre son el signo - delante para las que a la hora de realizar comprobaciones futuras, cualquier valor negativo haga aparecer el error, y después reaccionando en consecuencia según su valor. Estos datos se han extraído de una versión de RedHat 7.3 de Agosto del 2004 en el fichero `/usr/include/asm/errno.h` [15].

```
#define EPERM 1 /* Operation not permitted */
#define ENOENT 2 /* No such file or directory */
#define ESRCH 3 /* No such process */
#define EINTR 4 /* Interrupted system call */
#define EIO 5 /* I/O error */
#define ENXIO 6 /* No such device or address */
#define E2BIG 7 /* Arg list too long */
#define ENOEXEC 8 /* Exec format error */
#define EBADF 9 /* Bad file number */
#define ECHILD 10 /* No child processes */
#define EAGAIN 11 /* Try again */
#define ENOMEM 12 /* Out of memory */
#define EACCES 13 /* Permission denied */
#define EFAULT 14 /* Bad address */
#define ENOTBLK 15 /* Block device required */
#define EBUSY 16 /* Device or resource busy */
#define EEXIST 17 /* File exists */
#define EXDEV 18 /* Cross-device link */
#define ENODEV 19 /* No such device */
#define ENOTDIR 20 /* Not a directory */
#define EISDIR 21 /* Is a directory */
#define EINVAL 22 /* Invalid argument */
#define ENFILE 23 /* File table overflow */
#define EMFILE 24 /* Too many open files */
#define ENOTTY 25 /* Not a typewriter */
#define ETXTBSY 26 /* Text file busy */
#define EFBIG 27 /* File too large */
#define ENOSPC 28 /* No space left on device */
#define ESPIPE 29 /* Illegal seek */
#define EROFS 30 /* Read-only file system */
#define EMLINK 31 /* Too many links */
#define EPIPE 32 /* Broken pipe */
#define EDOM 33 /* Math argument out of domain of func */
#define ERANGE 34 /* Math result not representable */
#define EDEADLK 35 /* Resource deadlock would occur */
```

```
#define ENAMETOOLONG 36 /* File name too long */
#define ENOLCK 37 /* No record locks available */
#define ENOSYS 38 /* Function not implemented */
#define ENOTEMPTY 39 /* Directory not empty */
#define ELOOP 40 /* Too many symbolic links encountered */
#define EWOULDBLOCK EAGAIN /* Operation would block */
#define ENOMSG 42 /* No message of desired type */
#define EIDRM 43 /* Identifier removed */
#define ECHRNG 44 /* Channel number out of range */
#define EL2NSYNC 45 /* Level 2 not synchronized */
#define EL3HLT 46 /* Level 3 halted */
#define EL3RST 47 /* Level 3 reset */
#define ELNRNG 48 /* Link number out of range */
#define EUNATCH 49 /* Protocol driver not attached */
#define ENOCSI 50 /* No CSI structure available */
#define EL2HLT 51 /* Level 2 halted */
#define EBADE 52 /* Invalid exchange */
#define EBADR 53 /* Invalid request descriptor */
#define EXFULL 54 /* Exchange full */
#define ENOANO 55 /* No anode */
#define EBADRQC 56 /* Invalid request code */
#define EBADSLT 57 /* Invalid slot */
#define EDEADLOCK EDEADLK
#define EBFONT 59 /* Bad font file format */
#define ENOSTR 60 /* Device not a stream */
#define ENODATA 61 /* No data available */
#define ETIME 62 /* Timer expired */
#define ENOSR 63 /* Out of streams resources */
#define ENONET 64 /* Machine is not on the network */
#define ENOPKG 65 /* Package not installed */
#define EREMOTE 66 /* Object is remote */
#define ENOLINK 67 /* Link has been severed */
#define EADV 68 /* Advertise error */
#define ESRMNT 69 /* Srmount error */
#define ECOMM 70 /* Communication error on send */
#define EPROTO 71 /* Protocol error */
#define EMULTIHOP 72 /* Multihop attempted */
#define EDOTDOT 73 /* RFS specific error */
#define EBADMSG 74 /* Not a data message */
#define EOVERFLOW 75 /* Value too large for defined data type */
#define ENOTUNIQ 76 /* Name not unique on network */
#define EBADFD 77 /* File descriptor in bad state */
#define EREMCHG 78 /* Remote address changed */
#define ELIBACC 79 /* Can not access a needed shared library */
#define ELIBBAD 80 /* Accessing a corrupted shared library */
#define ELIBSCN 81 /* .lib section in a.out corrupted */
#define ELIBMAX 82 /* Attempting to link in too many shared libraries */
#define ELIBEXEC 83 /* Cannot exec a shared library directly */
#define EILSEQ 84 /* Illegal byte sequence */
#define ERESTART 85 /* Interrupted system call should be restarted */
#define ESTRPIPE 86 /* Streams pipe error */
#define EUSERS 87 /* Too many users */
#define ENOTSOCK 88 /* Socket operation on non-socket */
#define EDESTADDRREQ 89 /* Destination address required */
#define EMSGSIZE 90 /* Message too long */
#define EPROTOTYPE 91 /* Protocol wrong type for socket */
#define ENOPROTOOPT 92 /* Protocol not available */
#define EPROTONOSUPPORT 93 /* Protocol not supported */
#define ESOCKTNOSUPPORT 94 /* Socket type not supported */
```

```
#define EOPNOTSUPP 95 /* Operation not supported on transport endpoint */
#define EPFNOSUPPORT 96 /* Protocol family not supported */
#define EAFNOSUPPORT 97 /* Address family not supported by protocol */
#define EADDRINUSE 98 /* Address already in use */
#define EADDRNOTAVAIL 99 /* Cannot assign requested address */
#define ENETDOWN 100 /* Network is down */
#define ENETUNREACH 101 /* Network is unreachable */
#define ENETRESET 102 /* Network dropped connection because of reset */
#define ECONNABORTED 103 /* Software caused connection abort */
#define ECONNRESET 104 /* Connection reset by peer */
#define ENOBUFS 105 /* No buffer space available */
#define EISCONN 106 /* Transport endpoint is already connected */
#define ENOTCONN 107 /* Transport endpoint is not connected */
#define ESHUTDOWN 108 /* Cannot send after transport endpoint shutdown */
#define ETOOMANYREFS 109 /* Too many references: cannot splice */
#define ETIMEDOUT 110 /* Connection timed out */
#define ECONNREFUSED 111 /* Connection refused */
#define EHOSTDOWN 112 /* Host is down */
#define EHOSTUNREACH 113 /* No route to host */
#define EALREADY 114 /* Operation already in progress */
#define EINPROGRESS 115 /* Operation now in progress */
#define ESTALE 116 /* Stale NFS file handle */
#define EUCLEAN 117 /* Structure needs cleaning */
#define ENOTNAM 118 /* Not a XENIX named type file */
#define ENAVAIL 119 /* No XENIX semaphores available */
#define EISNAM 120 /* Is a named type file */
#define EREMOTEIO 121 /* Remote I/O error */
#define EDQUOT 122 /* Quota exceeded */

#define ENOMEDIUM 123 /* No medium found */
#define EMEDIUMTYPE 124 /* Wrong medium type */
```

**Código F.1:** Lista de los códigos de error documentados para el sistema Linux

# G

# Guía del desarrollador

Para continuar con el desarrollo del sistema `SecSys` se adjunta una pequeña guía de los puntos claves a conocer antes de intentar editar el código. Se dividirá en dos apartados cada uno correspondiente a cada uno de los softwares desarrollados en este trabajo.

## G.1. `SecSys Authenticator`

El programa consta de cinco ficheros de código C con sus respectivas bibliotecas:

1.– **checkpass.c:** Código básico del funcionamiento del programa.

2.– **shadow.c:** Código para el parseo del fichero de usuarios de Unix.

3.– **user_unix.c:** Código para el tratamiento de los datos de usuario.

4.– **errors.c:** Código para el informe de errores y su registro en el log propio de la aplicación.

5.– **types.h:** Definiciones útiles en las que se apoya el programa.

Los protitos de todas las funciones de estos ficheros se encuentran en el anexo C.

## G.2. Driver `SecSys`

1.– **SecSys.c:** Código principal del Driver.

1.1.– `printHeader`: Función para imprimir un mensaje característico en el log y ver de forma cómoda cuando comienza la ejecución del driver.

1.2.– `usb_secsys_register_device`: Función para el registro en el sistema de un nuevo dispositivo que será manejado por este driver.

1.3.– `usb_secsys_unregister_device`: Función para borrar el dispositivo del sistema una vez usado. Se debería ejecutar antes de extraer físicamente el medio.

1.4.– `printDiskInfo`: Función para imprimir los parámetros básicos de la unidad de almacenamiento que gestiona el driver.

1.5.– `secsys_init_secsys_drive`: Función con los comandos UFI para la obtención de información del estado de la memoria del dispositivo para poder comenzar a tratar los datos que contenga

1.6.– `secsys_blk_device_request`: Función para extraer de la cola del sistema de manera apropiada todas las peticiones que lleguen a la misma. Por cada comando extraído se llama a una de las funciones *transfer* para su procesamiento.

1.7.– `secsys_send_cmd`: Función para el encapsulamiento del envío de comandos al dispositivo. Solo precisa del comando ya construido para enviarlo al End-Point correspondiente.

1.8.– `secsys_get_file_system`: Función para obtener el sistema de ficheros leyéndolo de la memoria del dispositivo.

1.9.– `secsys_transfer2`: Función para el tratamiento de las órdenes del sistema.

1.10.– `secsys_transfer`: Función para el tratamiento de las órdenes del sistema.

1.11.– `init_buffer_pdu`: Función para inicializar la variable donde se almacena y construye un comando UFI.

1.12.– `extractInquiryData`: Función para la extracción de todos los valores devueltos por el comando Inquiry y su almacenamiento en una estructura de datos para su posterior uso.

1.13.– `printInquiryData`: Función para la impresión por el fichero de Log de todos los datos de la estructura Inquiry y comprobar su resultado.

1.14.– `printExt2Info`: Función para la impresión de los datos del sistema de ficheros Ext2 para su comprobación a través del log del núcleo.

1.15.– `readCredentials`: Función para la lectura de credenciales desde el fichero compartido con la aplicación `SecSys Authenticator`.

1.16.– `errorCodePDU`: Función para la localización de la causa de un posible error al enviar o recibir un comando UFI. Este valor es el devuelto por las funciones `usb_bulk_msg` o `usb_control_msg`.

2.– **files.c:** Código para la gestión de ficheros desde entorno de núcleo.

2.1.– `struct file* file_open`: Función para abrir un fichero dada su ubicación, sus opciones sobre el fichero y sus permisos.

2.2.– `void file_close`: Función para cerrar el acceso a un fichero.

2.3.– `int file_read`: Función para leer datos de un fichero dada su posición inicial, la variable donde se retornarán los datos, y la longitud que será leída.

2.4.– `int file_write`: Función para escribir datos de un fichero dada su posición inicial, la variable de donde se tomarán los datos, y la longitud que será escrita.

2.5.– `int file_sync`: Función para actualizar el descriptor de fichero para el manejo del mismo pos si hubiera sufrido algún cambio.